Master's Thesis

Master's degree in Industrial Engineering

Body Gestures recognition for Human Robot Interaction

MEMORY

June 29, 2022

Author: Joan Jaume Oliver Caraballo

Directors: Anaís Garrell Zulueta (Director)
Javier Laplaza Galindo (Co-Director)

Call: June 2022

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Abstract

In this project, a solution for human gesture classification is proposed. The solution uses a Deep Learning model and is meant to be useful for non-verbal communication between humans and robots.

The State-of-the-Art is researched in an effort to achieve a model ready to work with natural gestures without restrictions.

The research will focus on the creation of a temPoral bOdy geSTUre REcognition model (POSTURE) that can recognise continuous gestures performed in real-life situations. The suggested model takes into account spatial and temporal components so as to achieve the recognition of more natural and intuitive gestures.

In a first step, a framework extracts from all the images the corresponding landmarks for each of the body joints. Next, some data filtering techniques are applied with the aim of avoiding problems related with the data. Afterwards, the filtered data is input into an State-of-the-Art Neural Network. And finally, different neural network configurations and approaches are tested to find the optimal performance.

The obtained outcome shows the research has been done in the right track and how, despite of the dataset problems found, even better results can be achieved.

Acknowledgements

This project would not have been possible without the support of Anaís Garrell and Javier Laplaza. Many thanks to both of you, since in a way or another, you have been by my side, either giving tips and advises or helping me every time I got stuck.

Thanks for this opportunity and above all, for all the effort and personal involvement that you both have dedicated to me in a totally altruistic way.

And of course, to my family who have supported me all over this last years.

Joan Jaume Oliver
June, 2022

Acronyms

AAGCN Adaptive Graph Attention Networks. (33, 40)
ADAM Adaptative Moment Estimation. (34)
AGCB Adaptative Graph Convolutional Block. (33, 34)
AI Artificial Intelligence. (11, 25)
API Application Programming Interface. (46)

CNN Convolutional Neural Network. (9, 22)

DL Deep Learning. (i, 1, 4, 8, 9, 11, 15, 17, 29, 30, 33, 41, 42, 46, 53)

FPS Frames per Second. (39)

GAT Graph Attention Networks. (ix, 23, 33)
GCN Graph Convolutional Networks. (ix, 21–23, 33)
GNN Graph Neural Network. (ix, 20–23)
GPU Graphics Processing Unit. (49)

HRI Human-Robot Interaction. (3–7, 9, 47)

IDE Integrated Development Environment. (49)
IMU Inertial Measurement Units. (ix, 7)
IRI Insititut de Robòtica i Informàtica Industrial [25]. (1, 25, 28, 49)

ML Machine Learning. (1, 7–9, 11)
MPL Message Passing Layers. (20, 21, 24)
MUEI Master's Degree in Industrial Engineering. (45)

NLP Natural Language Processing. (45)
NN Neural Network. (i, ix, 8, 11–15, 17, 19–23, 25, 27, 31, 33–35, 37, 38, 42, 45, 49, 53)

POSTURE temPoral bOdy geSTUre REcognition. (i, x, 27, 35–39, 53)
PyG PyTorch-Geometric. (25, 46, 54)
PyGT PyTorch-Geometric-Temporal. (25, 46, 54)

ReLU Rectified Linear Unit. (33)
RGB Red-Green-Blue. (29)
RNN Recurrent Neural Network. (9, 17, 18)
ROI Region-Of-Interest. (27)

SOA State-of-the-Art. (i, 4, 6, 53)

TGNN Temporal Graph Neural Networks. (11, 23, 24)

Index

List of Figures

List of Tables

1 Preface

I have always loved challenges and tried to never give up, for me, this Master's Thesis has been an exhibition of self-improvement. Since a year ago, Deep Learning had been a totally new concept for me and my knowledge was nearly zero, so I did not expect to succeed in such a good way.

Right now, I have developed a model based on one state-of-the-art topic and feel comfortable with Deep Learning concepts. (*I can also follow more profoundly in papers, codes and conferences.*)

1.1 *Project origin and motivation*

During the months prior to the completion of this project, I was lucky enough to carry out the *Sistemes de Percepció* subject as part of the Master's degree. Although the topics discussed in the syllabus were quite basic, they were an introduction that helped me awaken my interest for Deep Learning.

There I also met Anaís Garrell, who gave the opportunity to work together on a more practical thesis in the Insititut de Robòtica i Informàtica Industrial [25] (IRI). For me, the idea of running a practical thesis was an opportunity I couldn't let go.

Up to that moment, I had only interacted with some basic Machine Learning algorithms (*KMeans, Decisions Trees, Logistic Regression, AdaBoost...*) and everything seemed scary. But now, thanks to this project I can continue filling my curiosity. And who knows, maybe change my professional career course.

2 Introduction

Although languages have become the primary communication tool, humans still heavily rely on nonverbal communication. Precisely, over the years, experts have determined that gestures and body language are the two main nonverbal modes humans use to exchange large amounts of information.

Fundamentally, gestures are actions done with the body that transmit specific messages. With the right gestures and body language, one can, for example, give an impression, offend someone or even get their attention.

At the beginning, the communication between humans and machines was mainly physical, mostly through keyboards or touch-screens. However, humans do not cooperate using touch-screens, their interaction requires the recognition and interpretation of speech, gesture or emotion. [44]

Human-Robot Interaction research focuses in a natural collaboration between humans and robots which is the key to make robotics effective enough to solve countless real-world challenges. [20]

Figure 1: BUDDY companion robot in elderly care. *Source:* [8]

Speech recognition is, then, one of the most convenient methods. Regardless of all the problems it has due to the huge variety of human accents or going out-of-control in noisy situations. Alternatively, vision based methods are one of the most popular fields of research, since they are also capable of expressing really complex information.

The main reason to pursue these touchless methods is to create an engagement between robots and humans, and thus, achieve a natural interaction between them. The better the collaboration, better job will the robots do. [33]

In the present thesis, the target is on the recognition of body gestures as a way to communicate with robots easily and in a natural way. With this in mind, a deep learning approach that is able to recognize body gestures in videos is proposed.

2.1 *Project Objectives*

Normally, knowing exactly the objectives to achieve and in consequence the steps to follow is a key factor in most projects. In this case, the project aims to reach an efficient Deep Learning algorithm that's able to recognise body gestures.

To do so, some sub-objectives must be fulfilled:

- Establish a feature extraction method that is able to find all the body joints.
- Research some current State-of-the-Art alternatives that might fit the problem in.
- Develop a Deep Learning model that uses a video as input and it is able to classify body gestures.
- Refine and research over the proposed model.

2.2 *Personal Objectives*

Sometimes, the objectives are not only from project side, but personal. In this case, the personal objectives, which have been crucial, are:

- Learn thoroughly Deep Learning, and be able to develop a model.
- Face difficulties in a totally new environment that forces to improve research tactics.
- Opening up doors and provide opportunities in a completely different professional area.

2.3 *Non-Objectives*

It is always important that when a project objectives are defined, they have to go with it's corresponding non-objectives.

In this case, the project only considers the implementation of a graph neural network for body gestures and not:

- All the task related to embedding the project inside IVO's[1] code.
- The interface that allows the robot to use these algorithms.
- A model with better accuracy results than State-of-the-Art alternatives.

2.4 *Scope of the project*

The main thesis scope defined by the director and co-director, can then be summarized as:

To research about Body Gestures recognition for Human-Robot Interaction and propose a solution for gesture classification using Deep Learning models, more specifically, the proposed algorithm should be based on graphs, use videos as inputs and apply spatial and temporal attention.

[1] IVO is one of the IRI [25] robots available to test Deep Learning algorithms with.

3 State of the Art

Nowadays, there has been a growing interest in making autonomous robots able to interact and communicate with people. This demanding objective has opened new research fields concerning the social abilities robots should acquire. Some examples of this are, for instance, assisting elderly people in homes or helping pedestrians in public spaces. Moreover, interest in collaborative robots for those examples requires safe physical interaction with humans.

Finding natural and efficient communication channels is essential in Human-Robot Interaction. If we take a look at the way humans communicate with each other, we see that about 70% of the communication is non-verbal communication. [46] Moreover, when humans want to communicate with other agents with whom they do not share a common spoken language –foreigners, babies or animals– most of the communication is non-verbal. [2, 23]

Figure 2: QT Robot: Robot For Autism and Other Special Needs Education. *Source:* [*38*]

3.1 *Human-Robot Interaction*

Although the notion of human interaction has been around as the notion of robots themselves, HRI is commonly referred to as a completely new emerging field. In its beginnings, questions about humans and robots like *How much will people trust robots?* or *What kind of relationship can a person have with a robot?* arise. These questions, decades ago were merely science fiction, but today, many of this issues are a reality of our society and have become core research fields. [4]

HRI is not the same as robotics, while robotics is focused on the creation of physical robots and the ways these robots need to manipulate their environment, HRI is concerned with how robots interact with people in society. Furthermore, HRI focuses on developing robots that can interact with people in various everyday situations, this opens lots of challenges, but at the same time it boosts its popularity.

Clearly, this social interaction can help improve as a society. As of today, the number of HRI applications are almost infinite and apply in most of the possible imaginable environments. For instance in the medical context, some of the most famous are elder care, companion, surgery assistance, autism or rehabilitation. [61]

At the beginning, HRI wasn't mature enough and was mainly dependent on humans that had to interact with third party elements like touch-screens, mouse or keyboards in order to cooperate with robots. Over the years and due to the impressive growth in the robotic industry new

algorithms based on different techniques like Computer Vision, have raised[2].

For now, most applications remain at the research stage, but this is expected to change rapidly. Since it is here that HRI has its major role to play, understanding how robots should behave around people and how people should respond and benefit from this interaction.

When humans cooperate among them, they do not use a keyboard and a mouse, they mainly interact with speech and gestures. Is it true, that with teleoperation humans can interact with robots, but if we want robots to be as good collaborators as other humans, they need to be able to reproduce common human-human interactions, including the abilities like recognizing intentions. [10, 3]

Due to the differences between how humans and robots communicate, current scope (in terms of natural language processing) is quite limited. In practice, it is difficult for robots to understand human interactions and properly express intentions, reason HRI community has put the focus on this communication barrier. [74]

When talking about overcoming this barrier, speech recognition is always part of the equation, but it has some problems related to human accents variety and noisy situations that rules them out. Is in these situations where methods like Vision become strong. Vision based technology for HRI is the touchless method able to express the most complex information. [11]

Although vision based methods seem to be the solution for HRI, they are not fully developed and some problems like moving backgrounds, or illumination conditions are yet to overcome. [19]

These touchless or empty-handed methods have received notable attention in the last years to make interactions more pleasing, but in general, touchless interaction present some advantages for communication between humans and robots:

- **Sterility**: In some environments, commonly medical, surfaces that are touched have to be made aseptic.
- **Vandalism**: There's situations, like in public spaces, that not having access to the devices, comes in handy.
- **Sharing**: When working with groups of persons, a touch screen could not be enough and instead, a single camera might be used to interact with everyone.
- **Fleeting**: In case of required short time responses, like in public transport or elevators we could benefit from this touchless interaction, just by moving our body.

If robots reach this new State-of-the-Art in which they can fully interact with humans, it would imply robots could be used in new domains on which they need to be closer to the user in order to help.

[2]This new algorithms allow contactless interactions due to being handled through external instruments like video camera recordings.

Figure 3: By raising the left arm, IVO is able to understand and turn to the left side.

3.2 *Bridging the gap between Machine Learning and HRI*

When it comes to communication with robots, it is possible to establish a set of gestures to communicate certain ideas in a similar way that gesture language works between humans. But, similarly to gesture language, this approach requires that both agents know which gestures compose the language and what meaning has each gesture. Otherwise, communication is not successful at all.

In order to improve Human-Robot Interaction (HRI), touchless technology is needed, to do so, Artificial Vision for face recognition or gestures detection among others is required. Is it true, that touchless interaction can also be achieved with some devices like Inertial Measurement Units (IMU) [65], but it creates a restriction and a dependency on a physical hardware, so in the end we would have not freed the user from a device usage.

Nevertheless, the fact of using specific wearables does not guarantee the touchless methods any advantages. Using gadgets on the human body takes back some of the touchless technologies strong points and also induces to a less natural communication.

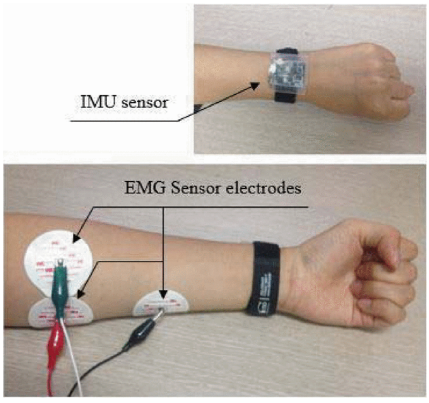


Figure 4: Example of an IMU device. *Source:* [65]

This explains the snowball in research on communication with the bare human body, dressed as one would dress to interact with another human, without adding extra instruments or devices on the user.

Machine Learning has been around for a few years now, and every day that goes on it grows more and more. As for example, for the new algorithm Dall·e 2 [14] that's able to generate realistic images and art from a description in natural language.

This growth, is today only possible thanks to the massive amount of data that's currently accessible. In the last decade, everyone has started using and bringing with them electronic devices like mobile phones. This devices, aside of what we usually use them for, generate huge amount of metrics that have proven be of benefits for Neural Networks.

Once Neural Networks started showing potential, Deep Learning algorithms started to spread over the industry, but despite their astonishing results, their acceptance was put under the spotlight. Basically, if a robot/algorithm is not reliable or safe it is not an option, independently of the price or productivity. [58]

This Machine Learning fever is also applicable in body language through Artificial Vision as we can see in:

- The 2021 Cruzcampo Lola Flores TV advertisement [13], in which they extracted morphological data from old videos.
- MediaPipe Pose [41], a ML solution for high-fidelity body pose tracking that can be used as sign language recognition, full-body control or gestures recognition

Vision algorithms are one of the most popular field of research in Machine Learning. Lately, it has been used to try and solve more quotidian problems while is still in some state-of-the-art development.

Research on gestures detection using NN has been vastly explored and it might seem it is reliable, but when it comes to a real-life working method, the temporal component supposes a big challenge.

As opposed to classic Vision algorithms, where the most popular topics are semantic segmentation or image classification, gestures detection implies a more complex and sophisticated algorithm. In order to fully understand body gestures, the algorithm first needs to extract the body information, then build the temporal graph sequence, understand it (*spatially and temporally*) and finally classify it.

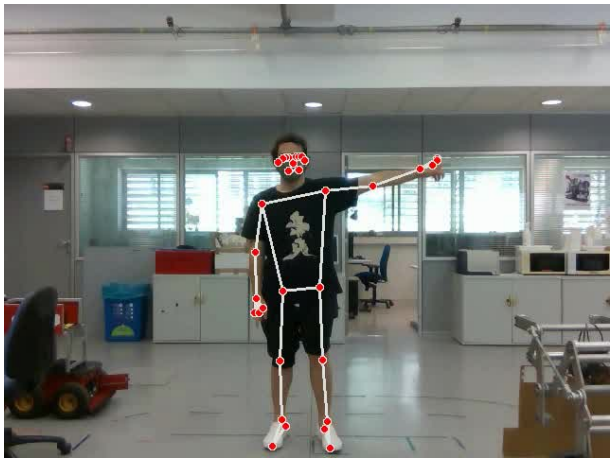


Figure 5: Pose extraction with Mediapipe.

3.3 *Body Gestures Recognition*

For HRI, one could have decided to use facial expressions or hand gestures recognition, but is it known that in order to achieve proper social interactions, body signals are also important, since in some cases, the body can even show better emotional expressions than faces [21], reason they need to be part of the big picture.

If we dig up a bit in the most common Machine Learning papers archives, we see there's quite an interest in understanding human activity and there's some active research in body gestures recognition that directly apply to HRI.

Conventional Deep Learning based methods manually structure the skeleton as a sequence of joint-coordinate vectors or as a pseudo-image, which is fed into Recurrent Neural Network (RNN) or Convolutional Neural Network (CNN).

However, new strategies that generalize convolution from image to graph, have been successfully adopted in many applications [29]. Firstly, they construct a spatial graph based on the natural connections of joints in the human body and then add the temporal edges between corresponding joints in consecutive frames.

In some papers like [6], the authors propose a two-step strategy, in which firstly they use a gesture detector to identify all possible gestures and afterwards if a gesture is found, they fed the information through a classifier that makes the prediction[3]. And in others like [18], the whole gesture prediction is done at once, joining different feature extraction algorithms.

Independently if the prediction is done in one or two stages, it's interesting to see that neither all papers focus on the same type body gesture communication, nor try to achieve the same results.

There's papers like [52] where the focus is about guessing where the human is pointing. Others work on identifying expressed emotions [45] and some of them even try to detect whether a human is willing to cooperate [32].

The approach used in this thesis, at least in terms of body analysis is similar to the one proposed in [42] where the relative position of hands and faces define the pose.

Within the Machine Learning community, different approaches for two stages gestures recognition based on graphs have appeared lately, among which, we can find:

- *ST-GCN*: Spatial Temporal Graph convolutional Networks for Skeleton-Based Action Recognition. [76]
- *STV-GCN*: Spatial Temporal Variation Graph convolutional Networks for Skeleton-Based Emotional Action Recognition. [69]
- *AS-GCN*: Actional-Structural Graph convolutional Networks for Skeleton-based Action Recognition. [34]

[3] This approach is quite similar to the one used in this Thesis as we can see in Section 5

- *RA-GCN*: Richly Activated Graph convolutional Network for Robust Skeleton-based Action Recognition. [66]
- *AAGCN*: Two-stream adaptive graph convolutional networks for skeleton based action recognition. [64]
- *DGNN*: Skeleton-based action recognition with directed graph neural networks. [63]
- *FGCN*: Feedback graph convolutional network for skeleton-based action recognition. [77]
- *Shift-GCN*: Skeleton-based action recognition with shift graph convolutional network. [12]
- *DSTA-Net*: Decoupled spatial-temporal attention network for skeleton-based action recognition. [62]
- *MS-G3D*: Disentangling and unifying graph convolutions for skeleton-based action recognition. [36]
- *PB-GCN*: Part-based Graph Convolutional Network for Action Recognition. [68]

On one hand, it's interesting to see that most of these algorithms are based on the same two datasets, NTU RGB+D [60] and NTU RGB+D 120[4] [35].

On the other, notice that all these algorithms use the two step strategy combined with graphs in which, firstly, the body is detected and afterwards, classified. They mainly use this structure, due to the strong adaptability to different circumstances the skeleton coordinates from the body provide.

[4]These two datasets were explicitly made to understand the 3D Human Activity.

4 Key Theory Concepts

This section presents and explains some basic theory concepts on Neural Networks that will be useful to properly follow the project afterwards. It starts from scratch, with the basic concept of Deep Learning and a one Neuron NN, and ends up detailing Temporal Graph Neural Networks with n number of hidden layers.

4.1 *Artificial Intelligence and Machine Learning (Deep Learning)*

When talking about Artificial Intelligence (AI), Machine Learning (ML) and Deep Learning (DL) concepts, it is easy to mix concepts.

On one hand, we have Artificial Intelligence, which is the broader concept of the three and can be understood as any technique that could help computers reproduce human intelligence.

On the other, there is Machine Learning, a subcategory of Artificial Intelligence which consists on improving tasks done by machines due to making them learn from data. Inside this group we can find, computer vision, social media algorithms or virtual voice assistants among others.

Then, we have Deep Learning, that is a subcategory of Machine Learning, mainly based on Neural Networks.

4.2 *Neural Networks*

Neural Networks (NN) can be understood as a mesh of elements (*neurons*) which interact among them in order to convert certain input into an output.

Let's imagine we are being asked to compare two flight prices: *We would basically say, the furthest the place we want to visit, the more expensive the flight would be.*

We are then assuming, that our output (*price*) is based on our input (*distance*):

$$price = m * distance + n \tag{1}$$

As for a one neuron network:

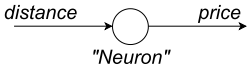


Figure 6: One Neuron example.

Thus, NNs need to be understood as complex structures that are capable of solving problems finding patterns in data.

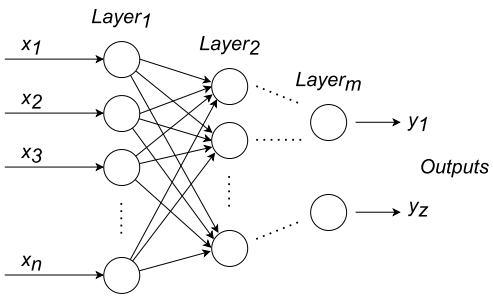


Figure 7: Complex Neural Network.

4.2.1 *Logistic Regression*

A Neural Network is a net of computational elements, *neurons*, each of them made up of combination of a linear function plus a non-linear one.

Let's take now as an example, a Neural Network that classifies dog images.

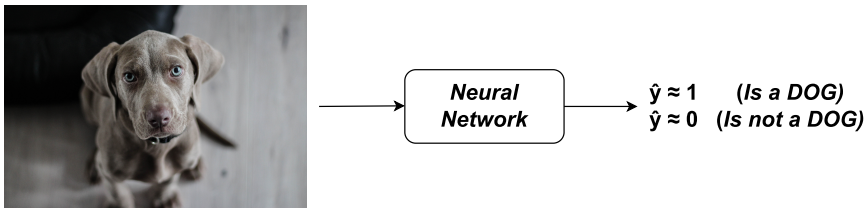


Figure 8: Dog Classification Neural Network

We have an image on which we extracted some features (x) that we want to use to predict the output probability in order to decide if the image is or not a dog.

This probability (z) can be then computed as follows:

$$z = \vec{w}' \cdot \vec{x} + b \tag{2}$$

In NNs $\vec{w}'$ (*weights*) and b (*bias*) variables are known as trainable parameters that are used in order to obtain the most possible accurate output.

But once we have an output value, we need to *activate* it in order to obtain a probability value bounded between $[0, 1]$. In this case, a non-linear function like *Sigmoid* can be used.

$$\hat{y} = Sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

$$\hat{y} = a = g(z) \tag{4}$$

These two equations together (Eq. 2 and Eq. 4) are known as Logistic Regression learning algorithm. This algorithm is typically used in binary output classification problems and its main objective is to get the minimum error between the data and predictions.

Although we have used $Sigmoid$ as *activation* function, it is not the only possibility. Depending on the needed output, different *activation* functions might be used.

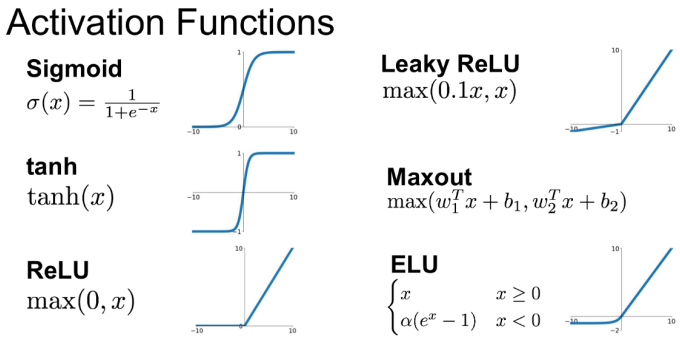


Figure 9: Common Activation Functions used in NNs. *Source:* [26]

4.2.2 *Loss Function*

The outputs obtained from NNs are predictions that need to be evaluated in order to say how good the performance of a network is.

The main idea is that our network should be able to output a value approximately equal to the one from the input data.

$$From : (x^{(1)}, y^{(1)}), ..., (x^{(m)}, y^{(m)}) \tag{5}$$

$$Reach : \hat{y}^{(i)} \approx y^{(i)} \tag{6}$$

In the Logistic Regression algorithm we have seen that in order to obtain the desired output, trainable parameters $\vec{w}'$ (*weights*) and b (*bias*) have to be taken into account.

$$\hat{y}^{(i)} = \sigma(w^T * x^{(i)} + b) \tag{7}$$

This parameters are the ones we want the NN to modify in order to achieve better outputs. But in order to know how to modify them, the difference between $\hat{y}$ and y, the net loss, needs to be minimized.

$$\mathscr{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2 \tag{8}$$

In order to compute the net loss, lot's of different functions like Eq. 8 could be used (*MAE, MSE, Hubber, Cross-Entropy...*). One of the most common Loss function is:

$$\mathscr{L}(\hat{y}, y) = -(y \cdot log(\hat{y}) + (1 - y) \cdot log(1 - \hat{y})) \tag{9}$$

Once a Loss function is defined, a new function J (cost) can be used in order to compute the loss over an m samples dataset:

$$J(w, b) = -\frac{1}{m} \sum_{i=1}^{m} \mathscr{L}(\hat{y}^{(i)}, y^{(i)}) \tag{10}$$

If then, we minimize this cost function using gradient descent, the trainable parameters values can be found.

In NNs terminology, this whole sequence is known as the *forward pass*.

4.2.3 *Gradient Descent Method*

Gradient Descent is an algorithm used to optimize the model finding the local minimum of a function. In Neural Networks Gradient Descent is used to optimize Cost Function (Eq. 10).

Step by step, the algorithm tries to achieve the local minimum of the function, minimizing its error.

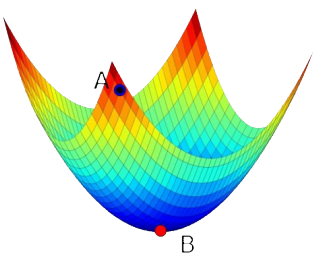


Figure 10: Simple Cost Function. *Source:* [75]

From the figure above, imagine our random initial guess it's at position A, in order to have the best possible output we need to iteratively get closer to the min, position B, in order to reduce Loss.

At each step of the learning algorithm, each trainable parameter is updated in order to get new results that are closer to the expected output. To do so, a new parameter (α) called learning rate is used.

The learning rate is used to specify how big or small the steps of each iteration are.

$$w = w - \alpha \cdot \frac{\partial J(w,b)}{\partial w} \tag{11}$$

$$b = b - \alpha \cdot \frac{\partial J(w,b)}{\partial b} \tag{12}$$

So in each $epoch$ (*NNs Step*) we will update the trainable parameters to minimize the NN output error.

4.2.4 *Backward Propagation*

In terms of NNs, *back propagation* is the process used in each step to update the trainable parameters.

If we have the following forward pass (*from Logistic Regression Algorithm*):

$$z = w^T * x + b \tag{13}$$

$$\hat{y} = a = \sigma(z) \tag{14}$$

$$\mathscr{L}(a,y) = -(y \cdot log(a) + (1-y) \cdot log(1-a)) \tag{15}$$

In order to update w and b, we need to follow inversely the fast forward process. So we will start computing the Loss function partial derivatives based on the output a and z.

$$\frac{\partial \mathscr{L}}{\partial a} = -\frac{y}{a} + \frac{1-y}{1-a} \tag{16}$$

$$\frac{\partial \mathscr{L}}{\partial z} = \frac{\partial \mathscr{L}}{\partial a}\frac{\partial a}{\partial z} = a - y \tag{17}$$

Once we have them, we want to finally compute the derivatives from both trainable parameters:

$$dw = \frac{\partial \mathscr{L}}{\partial w} = \frac{\partial \mathscr{L}}{\partial z}\frac{\partial z}{\partial w} = x(a-y) \tag{18}$$

$$db = \frac{\partial \mathscr{L}}{\partial b} = \frac{\partial \mathscr{L}}{\partial z}\frac{\partial z}{\partial b} = a - y \tag{19}$$

That can be later used together with the *Learning Rate* in order to update w and b. (Eq. 11 and Eq. 12).

We need to take into account that if our dataset contains m samples, the derivatives must be done based on the Cost Function and not the Loss. (Eq. 10)

4.3 *Shallow and Deep Neural Networks*

So far, we have only shown examples related to one neuron networks, but NNs are commonly more complex and normally have some concatenated layers.

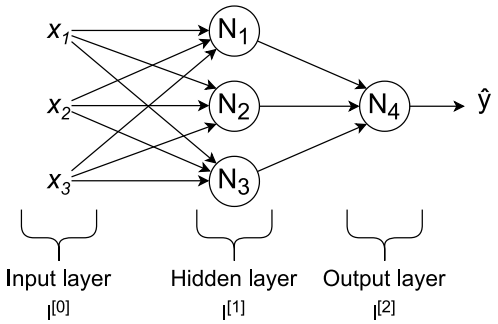


Figure 11: Shallow Neural Network example.

Mainly, all NN layers can be classified in three groups:

- Input Layer or layer zero, it's the first layer of the model and corresponds to where the data will be introduced.
- Hidden Layers, are used to define the complexity of the functions the network will be able to fit. In the Deep Learning community, if a NN has only one hidden layer, is known as a Shallow Neural Network, meanwhile if it has more than one, is known as a Deep Neural Network.
- Output Layer, is the last layer of the network and only contains the neuron that will compute the output value.

While thinking about the model that best fits your needs is interesting to point, that is generally better to fit the model increasing the layers, than increasing the number of neurons of some layers. Is it true that in both cases we are increasing the complexity the model can fit, but if we increase the neurons, we also increase the number of trainable parameters.

In Deep Neural Networks, defining the number of layers to use it is not an easy task. Each model has a different number of hidden layers since it always depends on the function you might want to fit. It's at the end an iterative task where lots of combinations are tested in order to see which works better.

4.4 *Vectorization*

When dealing with large number of neurons, applying the equations seen up to this moment, requires quite work and iteration[5]. In DL though, this gets simplified just by using *vectorization*.

Basically, instead of iterating over and over, we just stack equations in order to create matrices and use them instead.

If we get the example from Fig. 11, we see that the *hidden layer* is composed of three neurons,

[5]Iterate its computationally expensive and it ends affecting CPU performance.

so instead of having three equations to compute their corresponding output, we can vectorize it and do it all in one operation.

$$\begin{aligned} z_1^{[1]} &= w_1^{[1]T} \cdot x + b_1^{[1]} \\ z_2^{[1]} &= w_2^{[1]T} \cdot x + b_2^{[1]} \\ z_3^{[1]} &= w_3^{[1]T} \cdot x + b_3^{[1]} \end{aligned} \qquad \begin{bmatrix} z_1^{[1]T} \\ z_2^{[1]T} \\ z_3^{[1]T} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \end{bmatrix} \cdot X + \begin{bmatrix} b_1^{[1]T} \\ b_2^{[1]T} \\ b_3^{[1]T} \end{bmatrix} \tag{20}$$

$$Z^{[1]} = W^{[1]} \cdot X + b^{[1]} \tag{21}$$

Let's vectorize the whole forward and backward propagation equations for a Shallow Network:

Forward Propagation:

$$Z^{[1]} = W^{[1]} \cdot X + b^{[1]} \tag{22}$$

$$A^{[1]} = g^{[1]}(Z^{[1]}) = \sigma(Z^{[1]}) \tag{23}$$

$$Z^{[2]} = W^{[2]} \cdot A^{[1]} + b^{[2]} \tag{24}$$

$$\hat{Y} = A^{[2]} = g^{[2]}(Z^{[2]}) \tag{25}$$

Backward Propagation:

$$dZ^{[2]} = A^{[2]} - Y \tag{26}$$

$$dW^{[2]} = \frac{1}{m} dZ^{[2]} \cdot A^{[1]T} \tag{27}$$

$$db^{[2]} = \frac{1}{m} \sum_{i=1}^{n^{[2]}} dz^{[2](i)} \tag{28}$$

$$dZ^{[1]} = W^{[2]T} \cdot dZ^{[2]} * g^{[1]'} \cdot Z^{[1]} \tag{29}$$

$$dW^{[1]} = \frac{1}{m} dZ^{[1]} \cdot X^T \tag{30}$$

$$db^{[1]} = \frac{1}{m} \sum_{i=1}^{n^{[1]}} dz^{[1](i)} \tag{31}$$

Trainnable parameters

$$W^{[1]} = W^{[1]} - \alpha \cdot dW^{[1]} \tag{32}$$

$$b^{[1]} = b^{[1]} - \alpha \cdot db^{[1]} \tag{33}$$

$$W^{[2]} = W^{[2]} - \alpha \cdot dW^{[2]} \tag{34}$$

$$b^{[2]} = b^{[2]} - \alpha \cdot db^{[2]} \tag{35}$$

4.5 *Transformers*

In December 2017, a paper named *Attention Is All You Need* [70] revolutionized the Deep Learning world by proposing a totally new architecture that was dispensing recurrence and convolution, which at those moments, were fundamental pillars of most NNs.

4.5.1 *Recurrent Neural Networks*

Until 2017, *Recurrent Neural Networks* (RNN) were one of the main approaches used, they were used for speech recognition, voice recognition, time series prediction and natural language processing. The main idea of this architecture was to save information from specific layers in order to use it later.

Let's imagine we wanted to create a Neural Network that is able of analysing text sequences in order to classify peoples' opinions from web site reviews:

"It is what it says it is."

RNN architecture would grab the first word "*It*", analyse it, and use it as input for the second word "*is*". Basically the RNN is linking the previous word output to the next in the sentence.

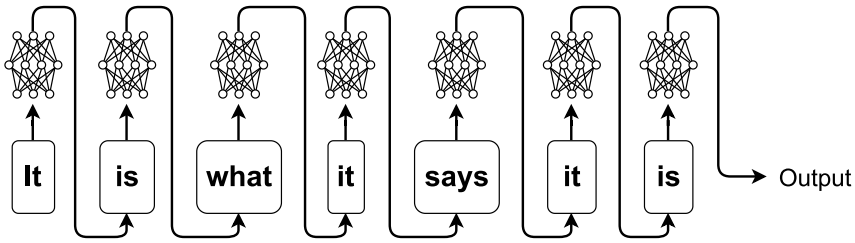


Figure 12: Recurrent Neural Network example.

Although this type of Networks have been used over the years, they face some problems:

- On one hand, we have what can be understood as a *lack of memory*. As humans, if we now have to remember the first word of this section we read, we might be in trouble. This same *lack of memory* happens to RNN, since as longer as the recurrent sequences are, the easier is some words importance might get lost once we reach the output.
- And on the other, the RNNs are unidirectional, so in some cases this could be tricky since it's not taking into account right-to-left relations.

4.5.2 *Attention*

In Deep Learning, *Attention* has a similar concept to the real word. Let's picture a situation where we are going to cross a street without a semaphore. As we have been taught, we will pay attention and take into account all inputs before crossing, this is the same in Deep Learning.

Let's suppose we are still working with the text sequence analyser from Fig. 12, but with a different sentence:

"This cow is not moving its tail"

In this sentence, due to the possessive "*its*", the word "*tail*" is related with the word "*cow*". If we had used RNN here, the bond between words could not be ideal.

But if we apply *attention* to it, we would have to grab each word independently and create a vector of how this word relates to the others:

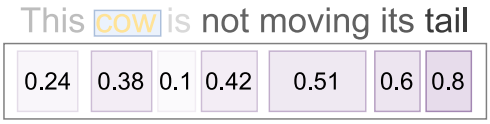


Figure 13: Attention key vector when *cow* is the query.

Basically, in the figure before, we have grabbed the word *cow* as query and obtained the key vector for it. (The influence each word has on *cow*).

We need then to compute the key vector $(A.K.V)$ for each word, and multiply each of them with their own *values*[6] before adding up. [30, 72, 71]

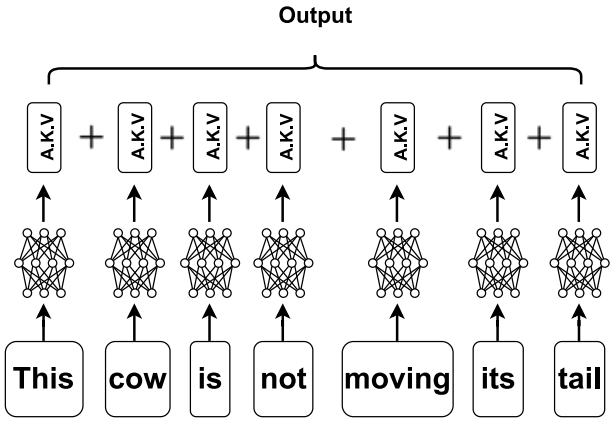


Figure 14: Attention example.

With *attention* we have then solved the main problem the RNN were facing, the *lack of memory*.

[6] As in RNN, is it interesting to notice that all the words go through a Neural Network in order to obtain a value.

4.6 *Graph Networks*

In the Neural Networks domain there's lots of types of networks and in consequence lots of different inputs (*audio, images, videos or data structures*). In Graph Networks, as its name states, graphs data structures are used as inputs. [59]

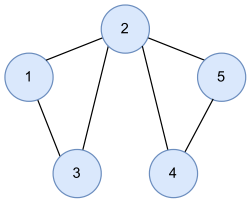
4.6.1 *Graphs Data Structures*

Graphs are easy to understand if we, for example, imagine them as *molecules* on which the elements (*nodes*) are interconnected (*through edges*). Or as people (*nodes*) relations (*edges*) in a Social Network.

Figure 15: Examples of Graph Neural Networks. *Source:* [*78*]

For a graph like the one on Fig. 15, each node can be described with a node feature vector. If the graph was a molecule, this feature vector would contain information about the atoms, but if the graph is a social network, it would contain the attributes about a person.

For instance, regarding the structural information of any graph, we have the adjacency matrix. This matrix is the key element that tells us which nodes are connected between them. [7]

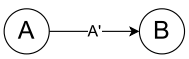


	1	**2**	**3**	**4**	**5**
1	0	1	1	0	0
2	1	0	1	1	1
3	1	1	0	0	0
4	0	1	0	0	1
5	0	1	0	1	0

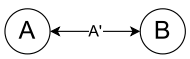
Figure 16: A five nodes graph with it's adjacency matrix.[8]

Graphs can be then classified based on their edges direction (*unidirectional/bidirectional*) and based on the number of nodes or edges (*static/dynamic*).

- *Unidirectional graphs*: Edges only have one direction.
- *Bidirectional graphs*: All nodes edges create relations in both directions.



(a) Unidirectional graph.



(b) Bidirectional graph.

Figure 17: Type of graphs based on their edges direction.

[7] It is a matrix of stacked vectors, for more information, go see Section 4.4
[8] The table is read from rows to columns and when an edge is present, it's marked with a 1

- *Static graphs*: These graphs are invariant and they do not change, neither over data nor over time. All graphs contain the same number of nodes and edges.
- *Dynamic graphs*: The number of nodes and edges, vary over data or time.

(a) Static graph. (b) Dynamic graph.

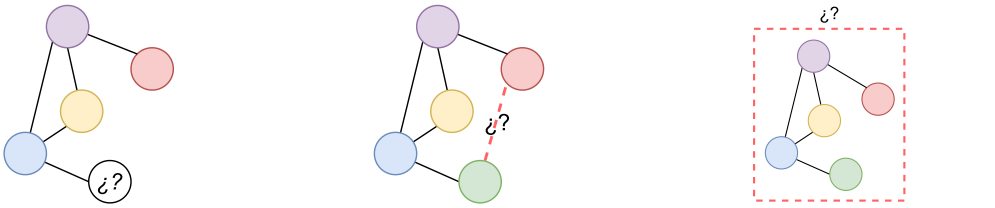
Figure 18: Type of graphs based on their edges or nodes number.

The main idea to use graphs is due to the fact that they not only contain data, but they also contain spatial relations between its entities, are permutation invariant, do not use euclidean space and size and shape do not affect them.

4.6.2 *Graph Neural Networks*

This type of Neural Networks operates on the graphs data structures and can work in some different ways:

- *Node-level predictions*, they can predict unknown node status through other nodes information. *E.g: We have a Molecule graph and we want to predict a property of an atom* (*node*).
- *Edge-level predictions*, can be used to predict new relations or unknown relations between nodes. *E.g: We have a Social Network graph and want to predict how much two different users communicate between them* (*edge*)
- *Graph-level predictions*, are typically used to predict completely new graphs. *E.g: We have an Skeleton based graph and we want to predict which gesture the graph represents*.



(a) Node-level predictions. (b) Edge-level predictions. (c) Graph-level predictions.

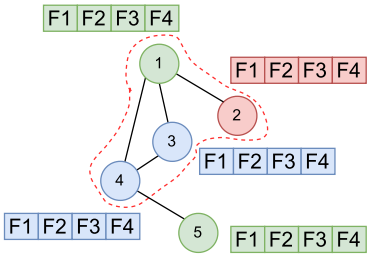
Figure 19: Types of Graph Neural Network model predictions.

As for this project scope, Graph-level predictions internally grab all the available information from their neighbours and fed it through *Message Passing Layers* (MPL) in order to construct a new feature vector that will contain knowledge in a compressed form. [9]

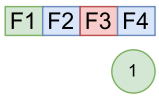
In a nutshell, the idea of MPL is a node, can send and receive messages along its connections with its neighbours so it can be later used.

[9] This process is done for all nodes simultaneously and in each GNN layer, in order words, MPL is applied in an iterative form, squeezing in each loop, more and more, the graph knowledge.

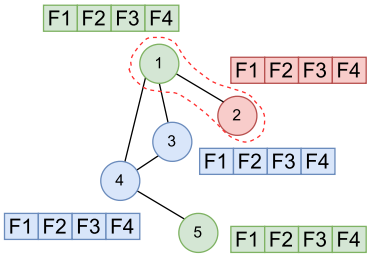
In Fig. 20 (a) and (b), we see how node 1 uses MPL in order to update himself with features from their neighbours (2, 3 and 4). While in Fig. 20 (c) and (d), node 2 can only update its own feature vector based on node 1.



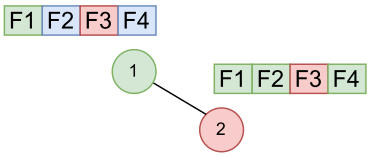
(a) Node 1 neighbours.



(b) Node 1 node embedding.



(c) Node 2 neighbours.



(d) Node 2 node embedding.

Figure 20: Message passing example for nodes 1 and 2.

This feature embedding that node 1 and 2 have done is commonly called *Node Embedding*.

Node embedding is a way to gather information of a specific node and its neighbours (performing *Message Passing Layers*) while also combining them (through an aggregation function like *average*). [16, 57, 31]

This process is similar to the convolution concept applied in images, where each pixel has information from its neighbours. Due to it, this process is also known as *graph convolution*.

As we understand from Fig. 20, the node embedding is computed step-by-step. So in the end, an exact same graph is obtained, but with different features. [10]

The message passing can be then defined as follows:

$$h_v^{k+1} = Update^k(h_v^k, Aggregate^k(h_v^k, \forall v \in \mathcal{N}(v))) \tag{36}$$

Where:

h_v:	Hidden state of node v
h_v^t:	Hidden state of node v at time step t.
$\mathcal{N}(v)$:	Neighborhood set of node v

In Eq. 36 we see the recurrent operators (*Update* and *Aggregate*) are missing from the equation, and this is because in GNNs there's lots of them. Additionally, each of this configuration, normally has a name to refer to, like it's the case of Graph Convolutional Networks.

[10] Depending on the desired deep of the NN layers, this *Message Passing* can be repeated.

The most common and typical operators to use are:

- *Update* Operator: $Mean$, Max, $NeuralNetwork$, $RecurrentNN$.
- *Aggregate* Operator: $Mean$, Max, $NormalizedSum$, $NeuralNetwork$.

For more information about possible recurrent operators to use, go see Annex A1.

4.6.3 *Graph Convolutional Networks*

Graph Convolutional Networks (GCN) are one of the most basic block in GNN, but at the same time, the most used. Over years, they were the only approach used to work with graph data.

GCN are a scalable approach for semi-supervised learning based on a variant of Convolutional Neural Networks (Convolutional Neural Network (CNN)). This type of networks are really useful due to the fact they scale linearly in the number of graph edges, while still learning hidden layers that encode graph structure and features. [29]

In GCN, node embeddings are computed as,

$$h_i^{'} = \sigma(\sum_{j \in N(i)} W * h_j) \tag{37}$$

This implies we need to multiply, the adjacency matrix, the features per node matrix and the learnable weight matrix.

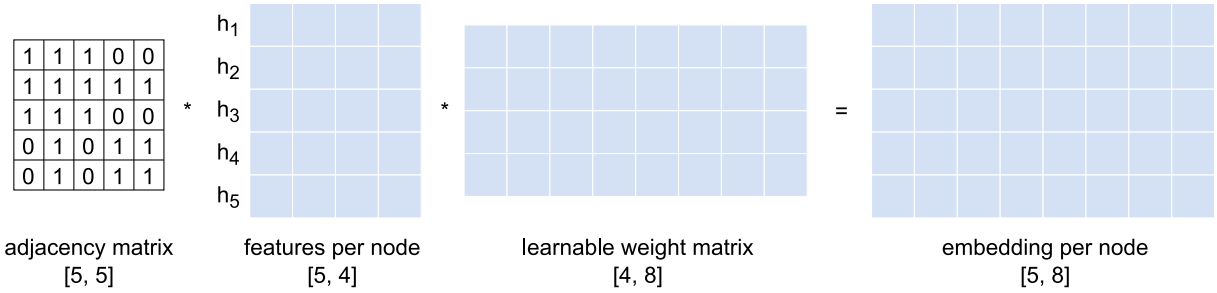


Figure 21: GCN node embedding example.[11]

- The adjacency matrix, is the responsible of containing all spatial relations of the graph and it's part of the input data.
- Features per node, is also part of input data, and are all the node feature vectors stacked.
- Weight matrix it's the unknown variable the NN will have to learn to obtain our desired outputs. (Based on the dimensions of the learnable weight matrix, we will obtain more or less outputs[12].)

[11] Embedding based on Fig. 16, with 4 features per node and 8 output classes.
[12] The weight matrix dimensions depend on each NN project particularities.

4.6.4 *Graph Attention Networks*

Graph Attention Networkss (GAT) [73, 15, 37, 7] are a subcategory of GNN that has revolutionized the Deep Learning community that focus on this type of data structures. They basically provide a new alternative to GCN that takes into account the influences between nodes.

In terms of GNN, GATs use the following specific recurrent operators.

- *Update* Operator:

$$\alpha_{vk} = \frac{exp(LeakyReLU(a^T[Wh_v||Wh_k]))}{\Sigma_{j\in\mathcal{N}_v} exp(LeakyReLU(a^T[Wh_v||Wh_j]))} \tag{38}$$

$$h^t_{\mathcal{N}(v)} = \sigma(\Sigma_{k\in\mathcal{N}_v}\alpha_{vk}Wh_k) \tag{39}$$

- *Aggregate* Operator:

$$h^t_v = h^t_{\mathcal{N}(v)} \tag{40}$$

GATs as their name states, are a type of Neural Network layers that merge graphs and attention [4.5.2]. The main idea of attention coefficient here, is quite similar to the approach we have already seen. In this case, instead of being applied to word sequences, is applied to the edges in order to see how much two nodes are related.

When using attention in a graph data structure, we can allow the network to identify those edges that are more important, and even exclude edges that were already part of the data but do not provide any useful information.

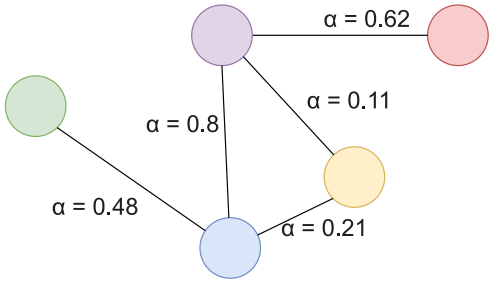


Figure 22: Attention coefficients example in a GAT

This new attention vector is then multiplied bt the adjacency matrix while computing the node embeddings. (In Fig. 21)

4.6.5 *Temporal Graph Neural Networks*

In some Neural Networks, like Temporal Graph Neural Networks (TGNN), input data is not only a graph, but a sequence of them.

We have to imagine TGNN inputs as a sequence of graphs that reproduce, for example, temporal evolution of peoples' movements.

In order to analyse this type of temporal data, we need to define:

- On one hand, a GNN model that will analyse each timestamp graph individually and calculate spatial embedding.

- On the other hand, a temporal model that will be used in order to calculate temporal embedding.

Figure 23: Sample from IRIGesture dataset.

Then, in the last graph of the whole sequence, we will have all the temporal and spatial information embedded.

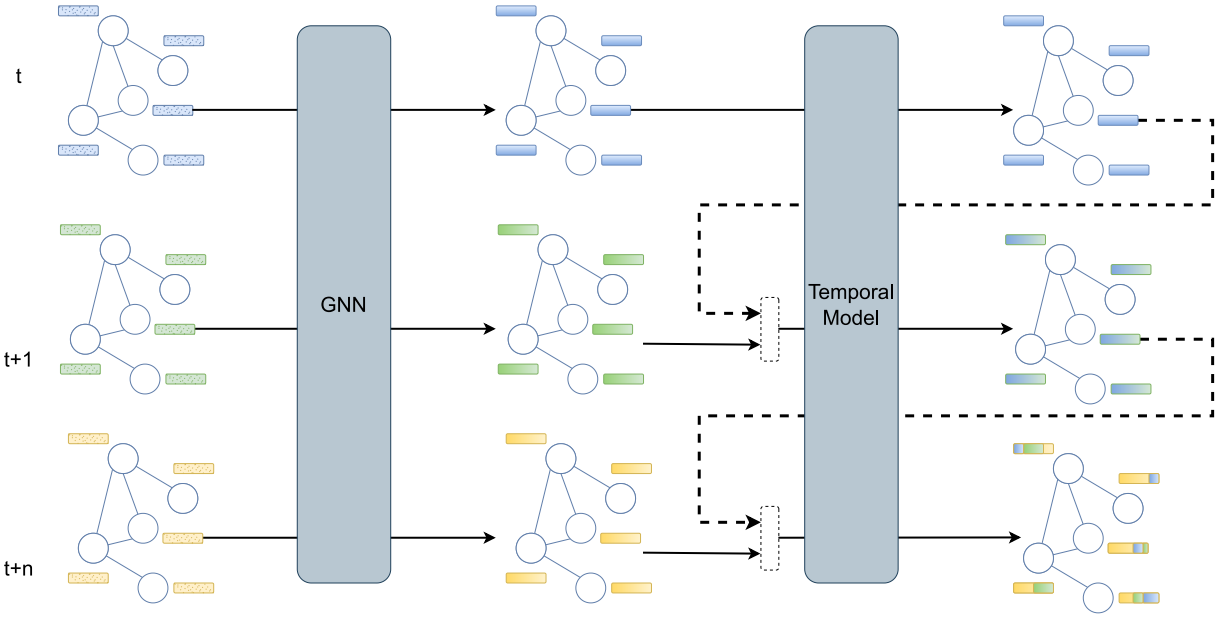


Figure 24: TGNN sequence example[13].

For more information about possible Spatiotemporal models (TGNN), go see Annex A2.

[13] In some Temporal models, there's also another *Message Passing Layers* used in order to feed the temporal embedding $(t-1)$ to the spatial ones before the graph model (t)

4.7 *Neural Network Framework: PyTorch*

Implementing a Neural Network from scratch is quite a difficult task due to the high complexity that's implied. Reason why, over the last years a few frameworks [14] have increased in popularity.

The most popular frameworks right now are PyTorch[49], TensorFlow [67] and Keras [28]:

Figure 25: Most popular DeepLearning frameworks.

The three of them are Python based frameworks that help with the Neural Network implementation. All of them provide *easy* ways to implement NNs just by defining the layers the user wants to use, while handling all the back propagation by themselves.

In this Thesis, PyTorch has been the one used, due to being the most popular among IRI AI researchers and also being the framework whose popularity has increased the most in the last years.

Besides the framework, a few interesting framework-libraries[15] have also been used:

- *PyTorch-Geometric* PyG is the most popular PyTorch library in order write and train neural networks that use graph data structures. [50]
- *PyTorch-Geometric-Temporal* PyGT is a PyG library extension that implements spatio-temporal signals. [51]

(a) PyTorch-Geometric

(b) PyTorch-Geometric-Temporal

Figure 26: PyTorch libraries used in order to interact with temporal graphs.

[14] A supporting structure around which something can be built.

[15] This libraries do not only provide a few functions to use, but define a complete way to interact, work and structure projects with graphs.

5 POSTURE Model: Method Development

This section discloses with a bit of detail the proposed tem**P**oral b**O**dy ge**STU**re **RE**cognition (POSTURE) model. Generally speaking, the model can be divided into two main areas. The first one is the responsible of the image treatment and body landmarks extraction, while the second one is the area in charge of the output prediction[16].

5.1 *System overview*

Given a video, our Neural Network is able to predict the gesture is being shown. Each video input is 30 frames long, from which we first extract and encode the position landmarks that are next fed to the NN.

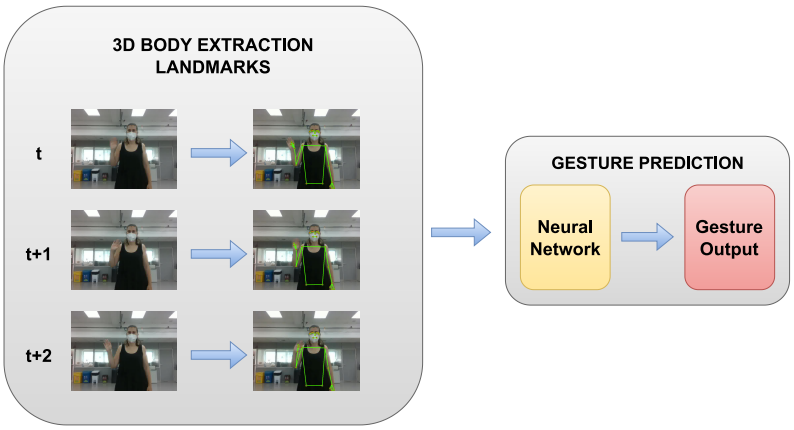


Figure 27: System Overview.

In this case, the model used is a classifier with a densely connected network that outputs a vector containing each gesture probability, that is used to make the final decision.

$$output = max(class_t(\%), class_{t+1}(\%), \quad ... \quad , class_n(\%)) \tag{41}$$

5.2 *Landmarks extraction*

In order to extract the desired landmarks or body joints, MediaPipe has been the tool used. MediaPipe Pose is a machine learning solution for high-fidelity body pose tracking[41]. This framework is really helpful since it's able to properly localize landmarks under different conditions of lighting, distance and half body configurations.

In order to get these joints, Pose uses a two-step strategy. First of all, the algorithm locates the Region-Of-Interest (ROI) within the image (*the body*). And afterwards, predicts the 33 landmarks using the ROI-cropped frame as input. In case of using videos as inputs, the system only tracks the body once and then the predicted pipeline get's the ROI from previous body positions.

This two-step algorithm, helps our prediction eluding the rotation, translation and scale problems while still granting a proper and stable output. A full body landmarks list is available in Fig. 28.

[16] All the code can be found either in Annex B or Github Repository. [47]

Pose outputs 33 3D landmarks for each video frame that we then use to build a 33 node unidirectional graph [4.6 Graph Networks].

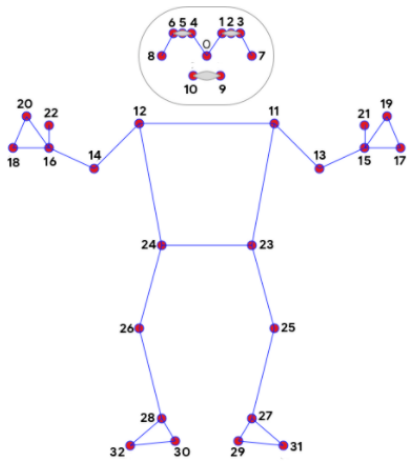


Figure 28: MediaPipe Pose Landmarks. *Source:* [*41*]

Each node is configured as a one dimensional vector with four elements, joint position x, joint position y, joint position z and joint *visibility*. Then, every one of these elements is a node feature.

$$11.left\_shoulder = (x, y, z, v) \tag{42}$$

5.3 *Body Gestures Dataset*

In order to train, test and validate the proposed model, IRIGesture dataset[55] has been used. This dataset is handmade and it was created in the IRI[25] laboratory.

The main feature of this gesture based communication dictionary is naturalness. The main objective is that everyone can communicate with robots and not only people who is already familiar with them.

The dataset contains 450 sample videos (divided in 10 different subjects[17]) of static and dynamic human gestures. Static gesture are those that require a certain amount of movement to be done and then they remain static. Dynamic gestures instead, are constantly in movement.

Static gestures

- **Attention**: Catch the robot's attention to give him an order.
- **Right**: Order the robot to turn right.
- **Left**: Order the robot to turn left.
- **Stop**: Order the robot to stop its trajectory.
- **Yes**: Approve a robot's information.
- **Shrug**: Inform the robot that you don't understand his information.
- **Random**: Random gesture, not necessarily a communicatve gesture.
- **Static**: Human is standing still.

[17] A wide range of users regarding age, gender, education level and culture was taken into account.

Dynamic gestures

- **Greeting**: Greet the robot.
- **Continue**: Order the robot to continue its path after telling him to stop.
- **Turn-back**: Order the robot to turn 180 degrees.
- **No**: Deny a robot's information.
- **Slowdown**: Order the robot to reduce its speed.
- **Come**: Order the robot to reach your position.
- **Back**: Order the robot to move back.

As for data recording, each human volunteer was recorded using an RGB camera. When human volunteers were asked to make a gesture they were provided with a vague explanation of the gesture intention. This was done to collect data that felt most natural to each volunteer. There was no restriction on which arms should be moved in each gesture whatsoever. Thus, different volunteers could make the same gesture in a very different way, using one arm or the other, or even both of them.

Each gesture was repeated three times, first 1 meter away from the camera, then 4 meters away and finally 6 meters away. Each video contains information of only one gesture, and all the videos were recorded indoors.

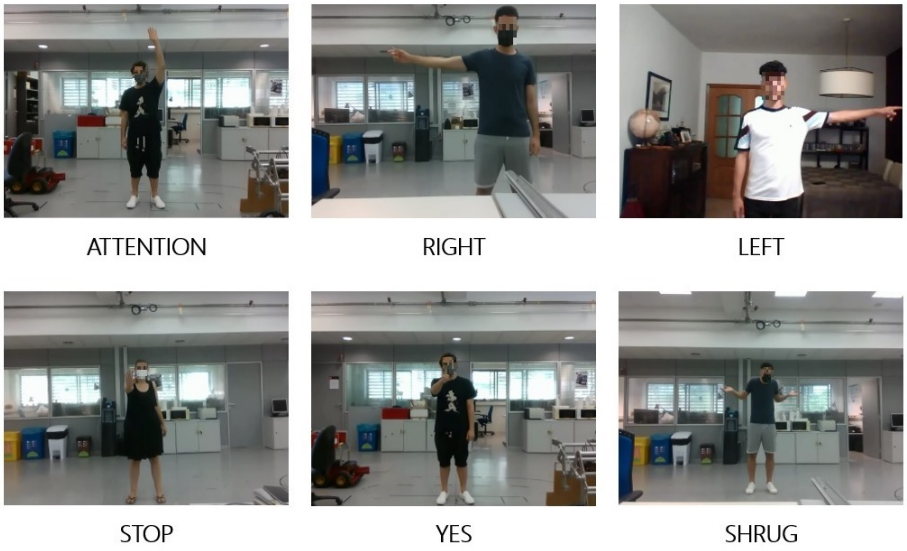


Figure 29: Some samples of static gestures recorded in the dataset.

5.3.1 *Data Augmentation*

In Deep Learning, having a large dataset for the model performance is essential. If we want our model to behave properly, it needs to be trained with a lot of data in order to properly understand all possible behaviours.

With only 450 videos (*30 per class*) we saw our model was having some problems understanding the data, so some Data Augmentation techniques were applied. Data augmentation is a way to artificially generate new data from already existing data.

Let's imagine we have a 19 frame video and we know that our model just needs 3 frames to work suitably. We could, for example, divide the whole frame sequence and obtain 6 videos.

$$n_{outputvideos} = total_{frames}//frames_{length} = 19//3 = 6 \tag{43}$$

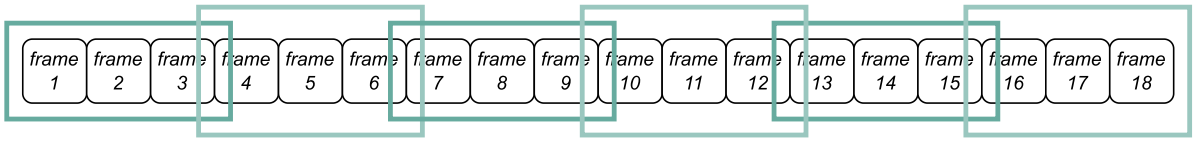


Figure 30: Example of data augmentation applied to videos.

Or we could try to even generate more data overlapping 3 frame sequences with a gap of 2.

$$n_{outputvideos} = (total_{frames} - frames_{length})//frames_{gap} = (19 - 3)//2 = 8 \tag{44}$$

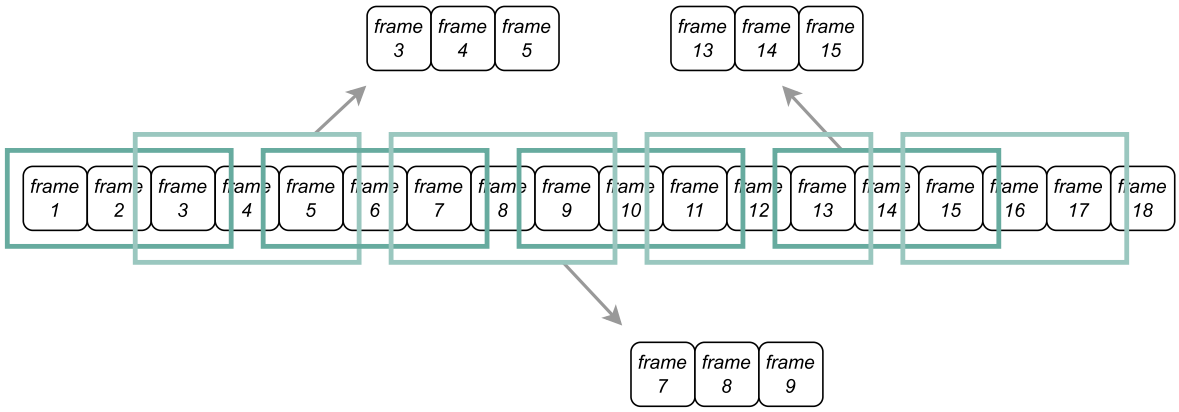


Figure 31: Example of advanced data augmentation applied to videos.

In the case of this dataset we have applied Data Augmentation with sequences of 30 frames and gaps of 3 frames generating nearly 3000 new videos.

5.3.2 *Train-Test Split*

With a dataset like ours, sometimes, general concepts about Train-Test splitting doesn't properly apply.

In Deep Learning, train and test are usually randomly split as $70/30$ or $80/20$. But there's cases in which this proportion doesn't make much sense [43], reason why we have decided to try and go for a $95/5$.

Just a 5% for a test dataset might not seem enough, but if we take into account that after data augmentation we have ~ 3000 input videos, a 5% means that ~ 150 videos are used to validate the model.

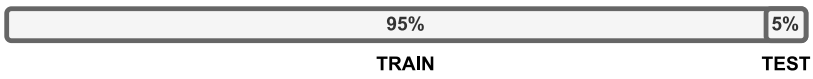


Figure 32: Dataset train/test proportions.

5.4 *Body Gestures Recognition*

The recognition of body gestures is accomplished by a NN that uses body joints (*landmarks*) extracted through Mediapipe Pose as inputs. (See section 5.2)

5.4.1 *Key Nodes Selection*

Out of the 33 possible joints that Mediapipe returns for each video frame, not all of them might come in handy.

If we take into account that most of our gestures exclusively involve hands and trunk, it was quite necessary to analyse the effect other joins like the ones from face, legs or feet were influencing our model.

In the course of time, the joints from face, legs or feet proved not to be worthy of being used and a filtering mask was finally applied. At the end, from the 33 initial joins, we ended up with 15.

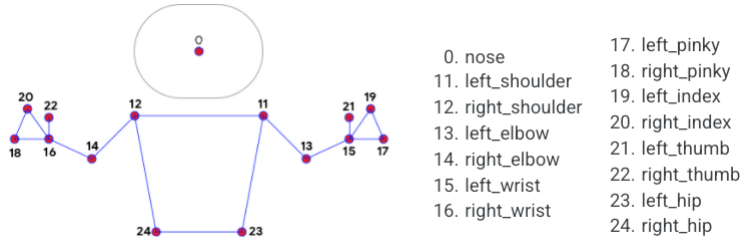


Figure 33: Masked Mediapipe landmarks. *Source:* [*41*]

5.4.2 *Gestures Selection*

As we have just seen, sometimes using all data is not the right choice and some data filtering might be needed.

During the first experimentation steps, we become aware that the model was somehow working for the train data but totally crashing and burning the test. This was due some poorly specific gestures like *random*.

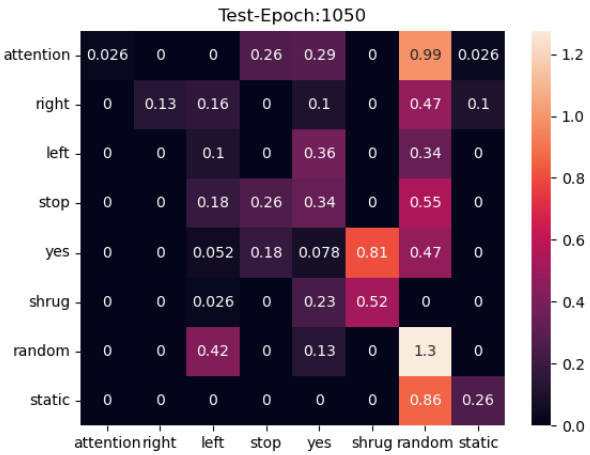


Figure 34: NN was mostly always predicting *random* when tested on Test dataset.

With this in mind, some gestures like *random*, *static* and *back* were finally excluded.

5.4.3 *Key Frames Selection*

Given the naturalness of the dataset, from the 450 original videos it has, none of them have the same length nor gesture speed. This implies that some gestures depending on the person who is doing them might start earlier or later, while others could be fast or slow.

Of all these issues, some are already solved by the Data Augmentation techniques used (section 5.3.1), but the problems that refer to the start gesture trigger are still present.

In some videos, gestures start at $0{,}5s$, meanwhile in others at $0.9s$, due to all this variation, we have set an offset and the first $0{,}7s$ $(15fps)$ of each video are ignored.

$$offset_{frames} = 0{,}7s * 22fps/s = 15fps \tag{45}$$

In this case, Eq. 44 changes to:

$$n_{outputvideos} = ((total_{frames} - offset_{frames}) - frames_{length})//frames_{gap} \tag{46}$$

5.4.4 *Class Imbalance*

At the beginning, we had the same number of videos per class, so our dataset classes were perfectly balanced.

Once we applied data augmentation techniques, the number of videos increased and the number of videos per gesture, due to different original video length, changed and dataset classes were no longer balanced.

For the sake of avoiding possible problems related to class imbalance, the gestures (%) have been taken into account while computing the loss and back propagation. (See section 4.2.2)

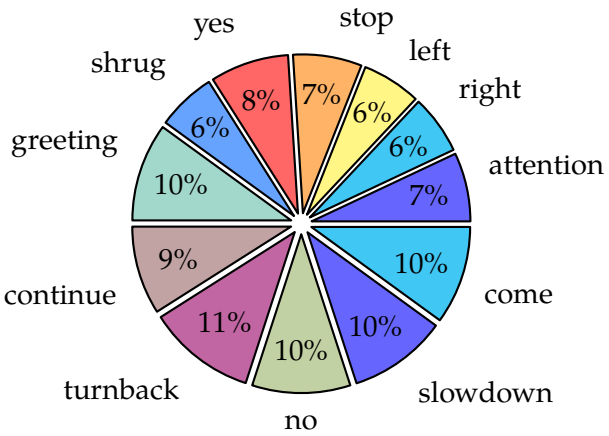


Figure 35: Class gestures (%) after Data Augmentation.

Basically, we have computed the class gestures (%) after Data Augmentation (See Fig. 35) and applied these factors to each related class. From the loss function point of view, this assignment implies the need of having to use these weights to penalize some most populated classes in order to force balance and avoid unintentional preference of some gestures above others.

5.4.5 *Hidden Layers*

Another interesting topic in Deep Learning is the number of hidden layers to use, but it's quite trial-and-error.

In this project, different combinations between 5 and 25 were tried and the magic number ended up being 15.

5.4.6 *Key Edges Selection*

Since we are going to apply spatial attention (See section 4.5.2), instead of providing a graph with edges as human body natural connections (See Fig. 33), we have created a highly connected graph on which all nodes are related with respect to all other nodes.

We do not want to fix and determine the main relations between nodes, but give total freedom to try and define them as the NN considers more appropriate.

5.4.7 *Neural Network Architecture*

Adaptive Graph Attention Networks (AAGCN)[64] are a State of the Art development that focuses on human body gestures recognition and it's based on *SpatioTemporal Graph Convolution Networks* [76].

The model uses Spatial and Temporal attention.(See section 4.6.5). (Encoded inside the *Adaptative Graph Convolutional Block* (AGCB)) (See Fig. 36).

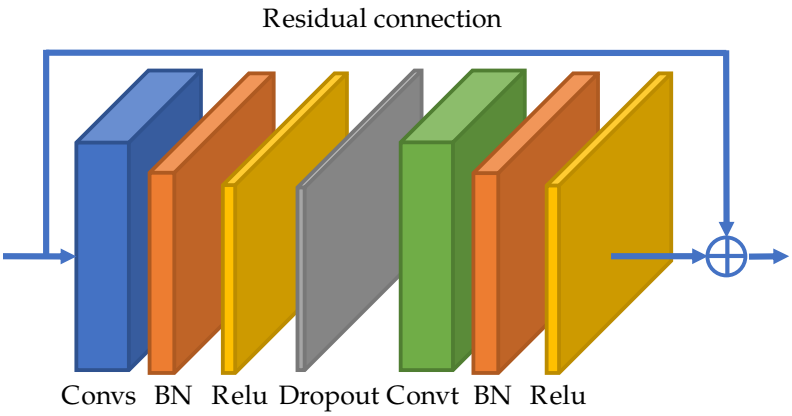


Figure 36: Adaptive graph convolutional block. *Source:* [64]

The AGCB layer is a quite complex block with:

- **Convs**: A GCN layer with Attention (GAT) applied over the spatial component. (See Section 4.6.4)
- **BN**: A Batch Normalization Layer used to achieve a faster learning rate. [5]
- **ReLU**: ReLU as the activation function (See Fig. 9)
- **Dropout**: A Dropout layer to prevent over-fitting. [9]
- **Convt**: A GCN layer with Attention applied over the temporal component.
- **Residual connection**: A connection that's added at the end of the block in order to keep the graph structure stable.

The proposed model (See Fig. 37) we have used the AGCB layer a total of 17 times combined with a few other blocks.

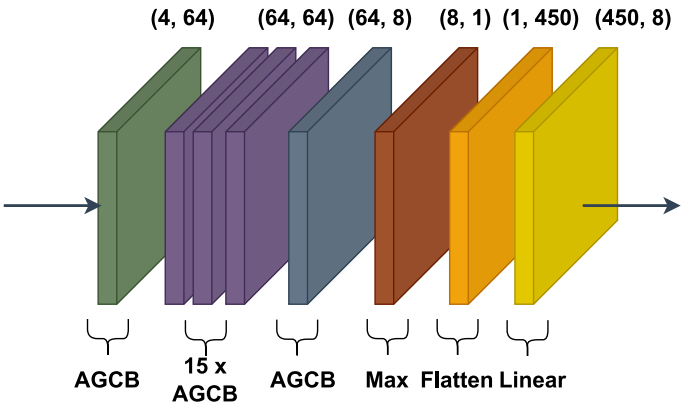


Figure 37: Proposed Model.

In the first AGCB layer, our model increases the number of features per graph from 4 to 64. Once we have 64 features per graph, we keep this number static and apply the Adaptative Graph Convolutional Block 15 times more. After we have had all these transformations, we decrease the number of features down to 8. With this, the model returns 8 features per node and frame, as if all of them were being classified individually.

Subsequently, we apply a max layer which, of all this features extracts the maximum values of each node and frame. Returning a unique graph that's a result of the mixture of the 8 graphs the last AGCB layer has returned.

With only one graph, we start the prediction part of the model, in which we flatten all features of all nodes and frames.

$$total_{features} = 1\frac{feature}{node} * 15\frac{nodes}{frame} * 30\frac{frames}{graph} = 450\frac{features}{graph} \tag{47}$$

With the 450 features in the same dimension, we apply a linear layer and reduce them to 8, as the number of gestures we want to classify.

As for the propagation itself, we have used CrossEntropy as Loss Function (See section 4.2.2) and Adam as Optimizer.

In terms of Loss Function, at least for classification projects, Cross Entropy is the most common function to use and it's one of the most important.

And in terms of NN optimizer, we have used Adaptative Moment Estimation (ADAM), since this optimizer involves a combination of two gradient descent methodologies, momentum and Root Mean Square Propagation. (See section 4.2.3).

Additionally to the optimizer we have also configured a scheduler that every 100 epochs reduces the Learning Rate by a 5% in order to force the NN achieve better results.

6 Experimentation and Results

In this section, all the refinement done to the POSTURE model for the purpose of achieving better performance[18] will be detailed.

6.1 *Model Experiments*

In stage zero of experimentation at the same time the model was being develop, some decisions like Key Nodes Selection (See section 5.4.1) were tested. Due to this, when the experimentation steps started those decisions were already clear and done.

Other decisions like the Key Frames Selection and Hidden Layers weren't that simple, so they required time and experiments.

- **Gestures Selection**

After having excluded the poorly specific and confusing gestures like *random*, *static* and *back* we started experimenting with the gestures in order to analyse their behaviour.

Experiment 1:
The first test done, was with 15 hidden layers and only the Static gestures (*attention*, *right*, *left*, *stop*, *yes* and *shrug*). With this combination, an accuracy about 82% against the test dataset (See section 5.3.2) was achieved.

Gestures	Accuracy
Static	82 %

Table 1: POSTURE results with Static gestures.

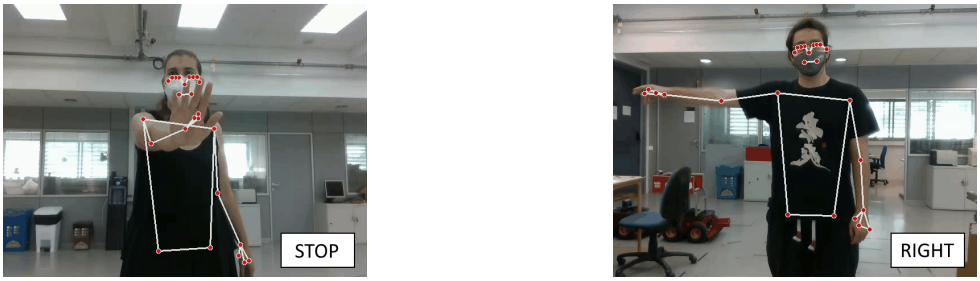


Figure 38: Predicted gestures with with Experiment 1 conditions.

Experiment 2:
With such an optimal algorithm, the next step in the list was to repeat the test, but with all the gestures instead. Dynamic (*greeting*, *continue*, *turnback*, *no*, *slowdown* and *come*) plus Static ones.

This second experiment, didn't go as good as we expected and the accuracy over the test dataset dropped nearly to 54%.

[18] Better performance needs to be understood as NN parameters optimization and model accuracy improvements.

Gestures	Accuracy
Static	82 %
All	54 %

Table 2: POSTURE results with Static and All gestures.

For some reason, when our model started to take into account dynamic and static gestures, it wasn't able to succeed with the training and perform properly.

Experiment 3:
Since the model didn't properly behave with all the gestures, the next step in the list was to test dynamic gestures alone.

With only the dynamic gestures, the model achieved a 62% accuracy, that was not as good as only with static gestures, but better than the global result.

Gestures	Accuracy
Static	82 %
Dynamic	62 %
All	54 %

Table 3: POSTURE results with Static, Dynamic and All gestures.

With this in mind, we can conclude that the model behaves better if the classifier is applied to dynamic and static gestures independently.[19]

	attention	right	left	stop	yes	shrug	greeting	continue	turnback	no	slowdown	come
attention	0,56	0	0	0	0	0	0,28	0	0	0	0	0
right	0	0,83	0	0	0	0	0	0	0	0	0	0
left	0	0	0,56	0	0	0	0	0	0	0	0,28	0
stop	0	0	0	0,42	0	0	0	0	0,21	0,21	0	0
yes	0	0	0	0	0	0	0,28	0	0	0,28	0	0,28
shrug	0	0	0	0,17	0	0,17	0	0	0	0	0,33	0,17
greeting	0	0	0	0	0	0	0,56	0,14	0	0	0	0,14
continue	0	0	0	0	0,42	0	0	0,42	0	0	0	0
turnback	0	0	0	0	0	0	0,12	0	0,48	0	0,12	0,12
no	0	0	0	0	0,14	0	0,07	0,07	0	0,49	0	0,07
slowdown	0	0	0	0	0	0,42	0	0	0	0	0	0,42
come	0	0	0	0	0	0	0	0,19	0	0,09	0,37	0,19

Table 4: Confusion matrix for All Gestures (*X: Prediction, Y: Reality*).

[19]Despite this, our intention is to continue testing both type of gestures and achieve a model can handle both of them independently.

If we take a look at the confusion matrix for all gestures (See Table 4) we see that the model works perfectly with some gestures like *attention right* or *left*, while it gets confused with others as *continue*, *come* or *slowdown*.

- **Hidden Layers**

Experiment 4:
Once the gestures to use were clear, the idea was to try and go deeper with the NN and see if increasing the number of layers, the accuracy would also increase.[20]

This experiment was done against the Static gestures and a number of 25 hidden layers were used, but the accuracy, instead of increasing, decreased a bit.

With this configuration, an accuracy of about 70% was obtained.

Hidden Layers	Accuracy
15	82 %
25	70 %

Table 5: POSTURE results with different number of hidden layers.

Up to this point, we concluded that increasing the number of hidden layers does not imply an increase in Test Accuracy.

- **Key Frames Selection**

Experiment 5:
After getting stuck trying to improve the model, both actively and passively, a small investigation about dataset quality cocked through. Mainly concluding, that not all the subjects nor videos had neither the same number of frames, nor the same gesture start point.

In some videos, the gesture started at $0{,}1sec$ while in others could start after the first second.

As a consequence, a time trigger of $0{,}7seconds$ (See Section 5.4.3) was applied and all the frames previous to this start time, were ignored.

Starting Time	Accuracy
0,0 sec	54 %
0,7 sec	60 %

Table 6: POSTURE results with initial frames filtering.

With this change, the accuracy with all the gestures increased up to 60%. So we ended up realizing that with this frame *filtering* was good for our model, since it cleared noise and disturbances.

[20] During the model developing state, a fixed number of 15 hidden layers were set.

- **Batch Size Selection**

Experiment 6:
Up to this point, the batch size of the NN wasn't really big, so just in case, an experiment increasing the batch size[21] was done, but the accuracy did not change.

In cases like this, is then better to keep the bigger batch size, since it smoothes the results.

Batch Size	Accuracy
32	60 %
128	60 %

Table 7: POSTURE results with different batch size values.

- **Dropout Layer**

Experiment 7:
After not succeeding, some more drastic decisions were made and a new Dropout layer with $0{,}2$ as scale factor to prevent over-fitting, was added to the model.

Unfortunately, the model accuracy totally crashed down to a 20% and this change was removed from the equation.

Dropout Scale Factor	Accuracy
0	60 %
0,2	20 %

Table 8: POSTURE results against a default Dropout Layer ($0{,}2$ *scale factor*).

Experiment 8:
Theoretically, when a Dropout layer is used, the algorithm should behave better, so in some cases is interesting to change and decrease the scale factor initially used in order to succeed.

Another small test was done, and the Dropout layer with a lower scale factor of $0{,}05$ was used, and even though the accuracy increased a bit against last trial, it was not enough at all.

The accuracy, just increased from $0{,}2\%$ to an insignificant $0{,}25\%$

Dropout Scale Factor	Accuracy
0	60 %
0,05	25 %
0,2	20 %

Table 9: POSTURE results against different Dropout factors.

[21] The batch size defines the number of samples that will be propagated through the network.

- **Class Imbalance**

Experiment 9:
After removing the Dropout layer and going back to the experiment number 6, the class imbalance strategy seen in Section 5.4.4 was tested.

Sadly, the accuracy kept steady at 60%.

6.2 *Model Results*

After all the experimentation done with the proposed model, a new investigative branch was opened in order to try to understand why the model was still stuck at 60% accuracy.

The main problem identified was related to the dataset used. After a few deep checks we realized the following:

- Not all subjects have been recorded with the same camera, and some of the videos have different number of frames per second (FPS) so the time difference between each graph in the temporal sequence is not constant.
- The length of the different gestures is not constant either, and while some gestures are done in less than a second, others might require a few more.
- Some of ours gestures like, *no*, *yes*, *come* or *slowdown*, have quite a dependency on the hands and finger positioning, but the landmarks extraction method used in this Thesis does not return many nodes per hand[22].

With all this in mind and despite of not being satisfied with this accuracy, we had to settle.

	Accuracy
Final Model (All)	60%
Dynamic	62 %
Static	82 %

Table 10: POSTURE final accuracy results.

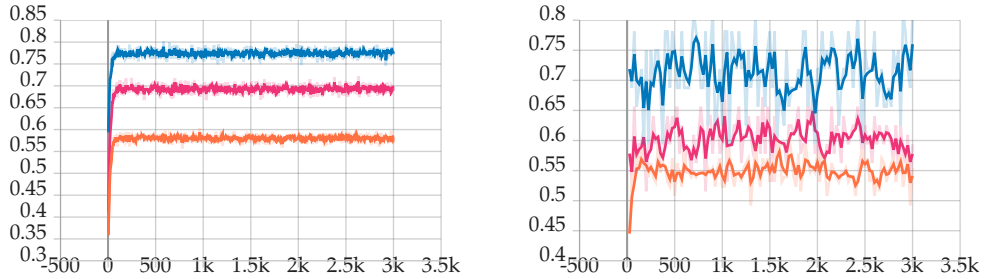


Figure 39: Train and Test accuracies over epochs for experiments 4 (*blue*), 5 (*pink*) and 9 (*orange*).

[22] At a late stage of the whole project, an alternative solution called Mediapipe Holistic [40], which improved Mediapipe Pose (method used), was found. But due to time limitations this could not be tested.

6.3 *State-of-the-Art Results*

The proposed model in this Thesis is a modification of an AAGCN (See section 5.4.7) done with the aim of working with our dataset (See section 5.3).

If instead of using our dataset, which as we have just seen has some problems, we could have used one that's not as natural as ours but is perfectly labeled and structured, and we would have achieved better results.

This dataset in question is NTU [35] & [60] and has been used by the AAGCN paper. In the paper, more than a 90% accuracy has been achieved. [64]

We can see that the model is able to predict gestures amazingly, so we could say that our main problem is on our data.

We tried to check out our model behaviour with the NTU dataset, but we find some incompatibilities:

- Our nodes have an extra feature (*visibility*).
- The number of nodes and position in the skeleton structure is different. (We have 15 nodes while NTU has 25).

These incompatibilities end up in having to change the number of layers the model has, and not being able to compare results nor easily proceed with fine-tuning[23].

[23] Fine-tuning is a process that takes an already trained model and tunes them to initialize a new model based on data from the same domain.

7 Project Plan

While working on a relatively big project, like Deep Learning one's, it is necessary to know how to face it. In many cases, if a project is not planed consciously it can be totally overwhelming.

Consequently, in this Master's Thesis a project plan methodology was defined from the very beginning.

7.1 *Project Plan Methodology*

Right now, the two most important project plan methodologies are *Waterfall* and *Agile*.

The first one, *Waterfall*, as its name states, it refers to the unidirectional water flow of a waterfall. This methodology sets, previous to the project start, all the phases that will make it up and the time to spend on each of them. It is commonly known as the fix, invariant and traditional method.

On the other hand, *Agile* is currently the most prevalent methodology and it is based on a non-sequential structure whose values are flexibility and adaptability. Instead of defining all the project phases, it divides the project into *Sprints*[24]. Once a *sprint* ends, all the developments achieved up to that point are used to create the next one.

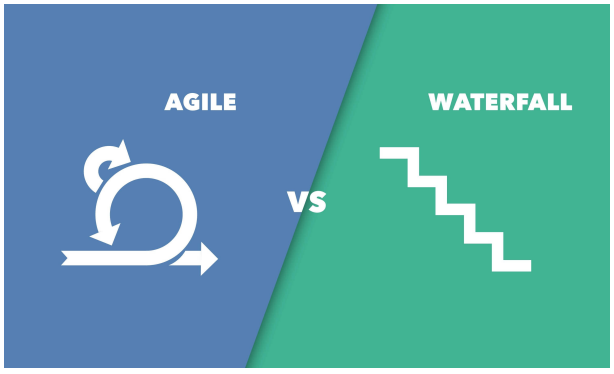


Figure 40: Project Plan Methodologies: Agile vs Waterfall. *Source:* [*1*]

The differences between both methodologies drive that only *Waterfall* can precisely define due dates, meanwhile *Agile* will only in the best cases, be able to approach them.

In this project, due to the fixed deadlines, a hybrid solution between both, that have provided flexibility, has been used.

Basically, intermediate due dates typically from *Waterfall* have been defined, keeping always in mind that they will suffer changes, get split or even discarded. This adaptability has been achieved with rigorous meetings where all steps of the project were reviewed.

[24] A sprint is a short, time-defined period that is used to complete a certain amount of work.

7.2 *Project Gantt*

In pursuance of a successful project, a project Gantt with some generic due dates, was used.

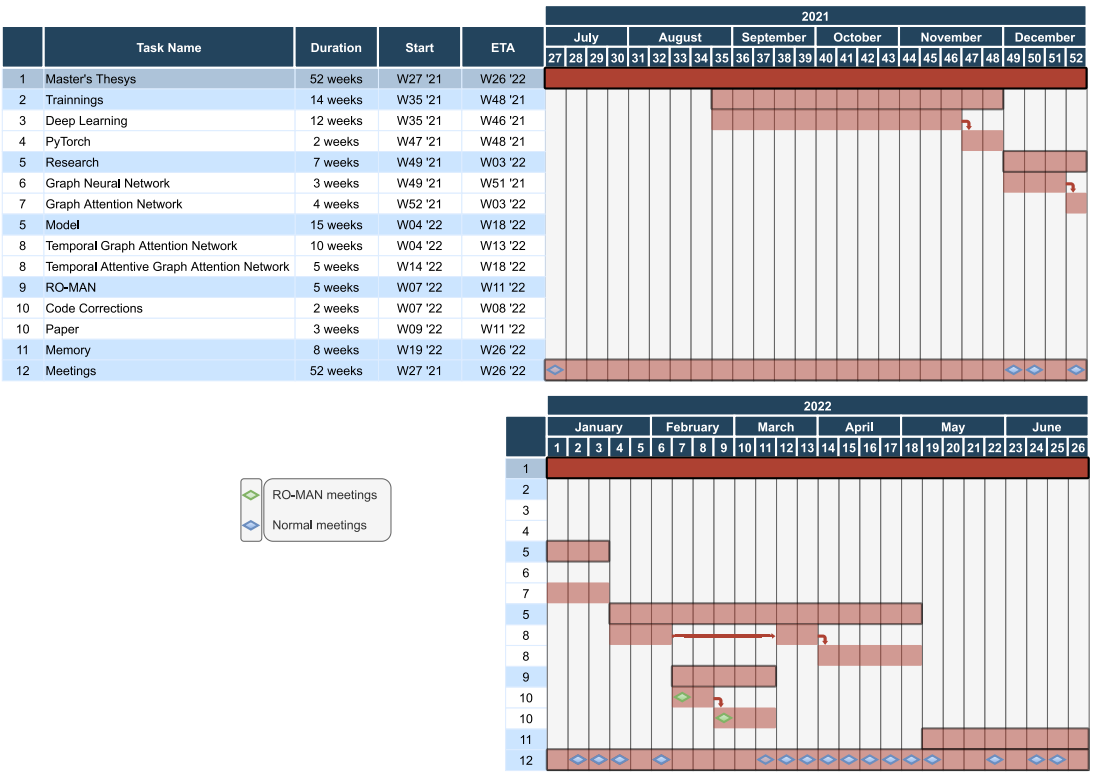


Figure 41: Project Gantt.

The main project was divided into 6 different *steps*:

- **Trainings**: Deep Learning material handled in this project is a bit out of the scope of the current Master's degree. In total, 14 weeks of trainings were required to start having some basic knowledge.
- **Research**: The topic handled is an state-of-the-art in the deep learning community and a 7 weeks of research (*Type of data, NN, etc.*) were done.
- **Model**: Once the ideas were clear, a NN had to be developed. A total of 15 weeks were spent in this step. Initially with the temporal approach and then with the temporal attentive approach.
- **RO-MAN**: In the middle of this thesis, the opportunity to write a paper together with the director and co-director was taken, and the rest of the project was paused for 5 weeks.
- **Memory**: A total of 8 weeks were saved to write the whole project memory.
- **Meetings**: During the whole 52 weeks, continuous meetings with the directors have been done.

This Gantt has been a key factor to achieve the desired target date and finish the whole project in time.

7.3 *Trello*

In a complementary way, a *Trello* to organize the whole project has also been used. Trello is a web-based Kanban-style, powerful list-making application.

Trello is also a visual tool that's used for organizing work, it is also customizable and flexible so it easily adapts to the needs.

Basically, it allows to:

- Go from idea to action in seconds by creating cards and dragging them across the board to follow the project's progress.
- Add checklists, labels, and due dates the way you see fit and give your projects the fuel they need to get across the finish line.
- Free your brain from remembering every planning detail.

Trello cards are the key to organized work where every single part of the project can be managed, tracked, and shared. (These cards can contain an ecosystem of checklists, due dates, attachments, conversations, and more.)

In this *Trello*, an *Agile* focus has been used (small tasks with achievable outputs that are constantly in revision), and all the *Agile Sprints* can be found as cards from the meetings list, and the steps done are available in the Steps list.

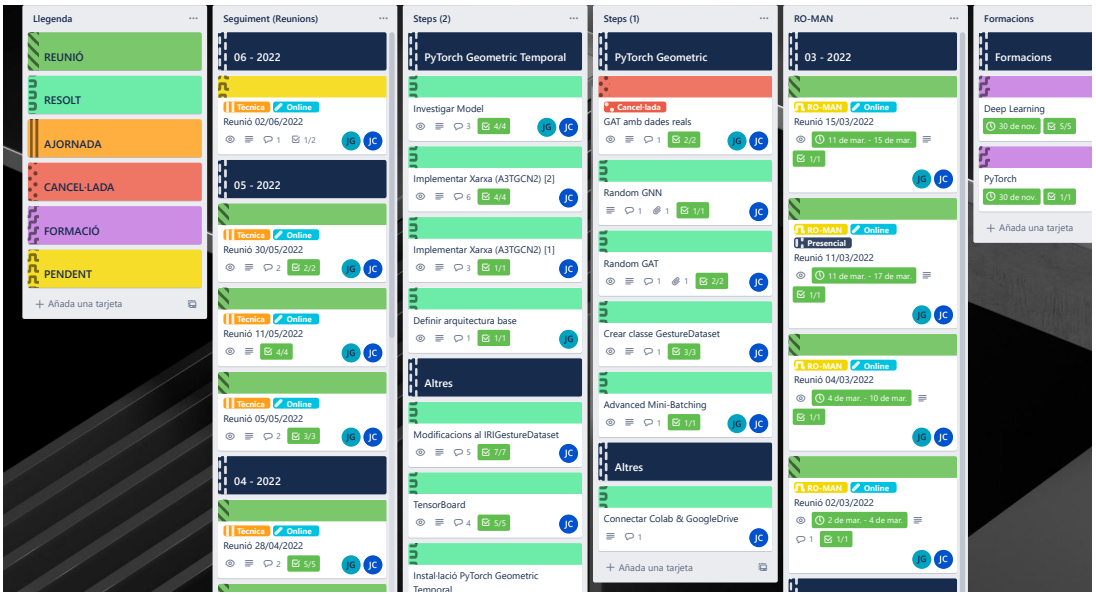


Figure 42: Project Trello at June 3, 2022[25].

[25]This tool has been for personal use only and the lists are in Catalan.

8 Trainings

The theory used in this Master Thesis that has allowed having a working NN model is not part of the current Master's Degree in Industrial Engineering (MUEI) curriculum [39]. In pursuance of achieving a working NN, some trainings were done.

8.1 *Deep Learning Specialization*

The first training was **Deep Learning Specialization** from Coursera and DeepLearning.AI [43], this specialization starts from scratch and gets harder each course, up to the point in the last weeks you are required to build a few NN with TensorFlow [67].

The whole program, it is divided in 5 different courses and has been thought out to be done in $\approx 180h$.

Broadly speaking, the course syllabus are:

- **Course 1: Neural Networks and Deep Learning**. In which the foundational concept of neural networks and deep learning were studied.
- **Course 2: Improving Deep Neural Networks: Hyperparameter Tuning, Regularization and Optimization**. Opens the deep learning black box to understand the processes that drive performance and generate good results systematically.
- **Course 3: Structuring Machine Learning Projects**. Teaches how to build a successful machine learning project and get to practice decision-making as a machine learning project leader.
- **Course 4: Convolutional Neural Networks**. Teaches how computer vision has evolved and helps you become familiar with its exciting applications such as autonomous driving, face recognition, reading radiology images, and more.
- **Course 5: Sequence Models**. Helps being familiar with sequence models and their exciting applications such as speech recognition, music synthesis, chatbots, machine translation, Natural Language Processing (NLP), and more.

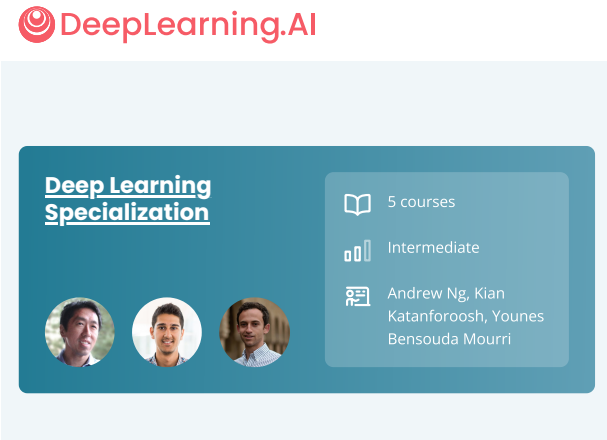


Figure 43: Deep Learning Specialization Courses. *Source:* [43]

8.2 *PyTorch Basics*

Once the basic knowledge from Deep Learning was studied, it was time to start learning about the framework to use. For that reason, the second training done was the **PyTorch Basics** [49].

This training can be done following the literature and can be run in the cloud (Colab)[22] or locally.

It is mainly divided into the following topics:

- **Quickstart**. To familiarize with PyTorch's Application Programming Interface (API).
- **Tensors**
- **Datasets and DataLoaders**
- **Transforms**
- **Build Model**
- **Automatic Differentiation**
- **Optimization Loop**
- **Save, Load and Use Model**

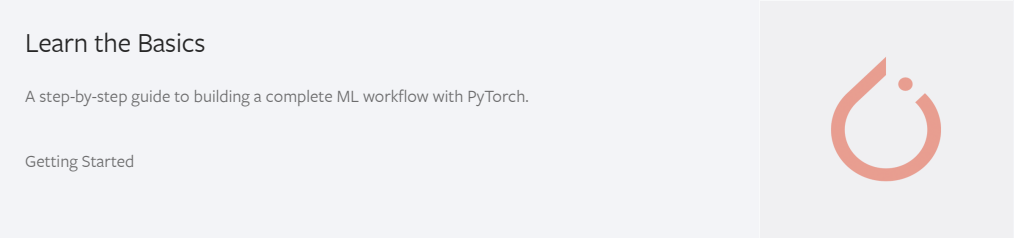


Figure 44: PyTorch Basics. *Source:* [*49*]

8.3 *PyTorch-Geometric & PyTorch-Geometric-Temporal*

And finally, since two framework-libraries (PyG and PyGT) have also been used, they have also been investigated in depth.

In the case of **PyG**, the Colab and video examples from the main webpage were taken into consideration.

- **Introduction: Hands-on Graph Neural Networks**
- **Node Classification with Graph Neural Networks**
- **Graph Classification with Graph Neural Networks**
- **Graph Attention Networks (GATs)**

And for **PyGT**, since it's been the main key library-framework, the whole library documentation was read and investigated.

9 RO-MAN Conference

RO-MAN 2022 is the 31st IEEE International Conference on Robot & Human Interactive Communication. [53]

This conference is a well-known forum with state-of-the-art discussions that covers a lot of topics related HRI. This year it will take place in Naples, Italy and anyone can land a paper.

Writing a paper for a conference wasn't initially in the plans of this thesis, but the opportunity raised and together with the director and co-director a paper called *Body Gesture Recognition to Control a Social Robot* was done and sent.

The paper was based on the IRIGesture dataset (See section 5.3 and a previous student Master's Thesis that focused on Static Body Gestures. [54]

All the work related to the paper was divided into two main topics, code review, fixes and improvements on one hand, and on the other, the whole writing itself.

- The first step done was cloning the previous student GitHub Repository [27] and try to execute its code. At the beginning, the code wasn't working for us, and some bug-fixes and code corrections had to be done [26]. But after a few weeks of work, the code achieved better results than expected.

- The paper writing was done once the code was working, and we ended up with a 6 page paper.

The 25th of March, the paper was handed over as the 317 RO-MAN 2022 submission. [33] And although we expected the paper to be accepted, on the 31th of May we were told that only 238 papers could be accepted and ours wasn't one of them[27].

[26] All the changes done to the code in order to work were pushed and can be seen, read and commented from the GitHub commits history.

[27] Despite the fact the paper was not accepted, we are planning to make some minor corrections and deliver it on other conferences like Humanoids 2022. [24]

10 Budget

In order to compute the total amount of money that is required to reproduce this project, the costs have been divided in a few different categories (hardware, electricity, licenses and trainings).

- **Hardware**

We need to take into account the two computers used. On one hand, a personal computer (PC) and on the other, Tyan, an IRI Neural Network GPU server with 10 NVIDIA GeForce RTX 2080Ti.

As for Tyan, since one GPU at a time was used, only the price of a computer with one 2080Ti will be taken into account. In terms of money, the personal computer is about $750€$ and the Tyan alternative, $2.200€$

$$Hardware = PC + Tyan_{Alternative} = 750€ + 2.200€ = 2.950€ \tag{48}$$

- **Electricity**

In case of consumed electricity, we need to take into account that, a personal computer consumes $\approx 80\ watt$ and the Tyan alternative $\approx 150\ watts$. If we then consider that Tyan has been running a total of $120h$ and the personal computer about $1.080h$:

$$PC_{Electricity} = N_{Hours} * Power = 1.080h * 80w = 86{,}4kwh \tag{49}$$

$$Tyan_{Electricity} = 120h * 150w = 18kwh \tag{50}$$

$$Total_{Electricity} = (86{,}4kwh + 18kwh) * 0{,}25€/kwh = 26{,}1€ \tag{51}$$

We have taken $0{,}25€kwh$ as electricity average price in Spain.

- **Licenses**

For this project PyCharm Professional [48] has been the IDE used to write and test all the code.

$$PyCharm_{Professional} = 89€/year * 1year = 89€ \tag{52}$$

- **Trainings**

Of all the trainings done, although all can be done for free, the Deep Learning one has a subscription mode that enables some really helpful exercises.

$$DeepLearning_{Specialization} = 49€/month * 4months = 196€ \tag{53}$$

- **Labor**

In terms of labor, it's important to take into consideration the hours the student and co-director have spent.

$$Engineer_{Student} = 40€/h * 1200h = 48.000€ \tag{54}$$

$$Engineer_{Co-Director} = 50€/h * 150h = 6.000€ \tag{55}$$

$$Labor = Engineer_{Student} + Engineer_{Co-Director} = 48.000€ + 6.000€ = 54.000€ \tag{56}$$

- **Total Cost**

If we then add up all the costs, we see the total project cost is around $57.000€$.

$$Total_{Cost} = Hardware + Electricity + Licenses + Trainings + Labor \tag{57}$$

$$Total_{Cost} = 2.950€ + 26{,}1€ + 89€ + 196€ + 54.000€ = 57.261{,}1€ \tag{58}$$

11 Environmental Impact

As the project has been entirely developed only implementing and using software, the real impact it had on the environment is on its electrical power consumption.

ElectricityMap[17] is a webpage who provides data quantifying how carbon intensive electricity is on an hourly basis across 50+ countries.

Regarding the data from ElectricityMap, the carbon emissions from 2021 range from $164{,}21$ gCO_2/kwh to $337{,}40$ gCO_2/kwh with an average value of $167{,}66$ gCO_2/kwh. (See Fig. 45).

And based on the power consumption already specified in the Budget (See section 10):

$$Carbon_{emissions} = 160 gCo_2/kwh * 104{,}4 kwh = 16{,}70 kgCO_2 \tag{59}$$

We can compute whole project carbon emissions, that are about $16{,}70$ $kgCo_2$

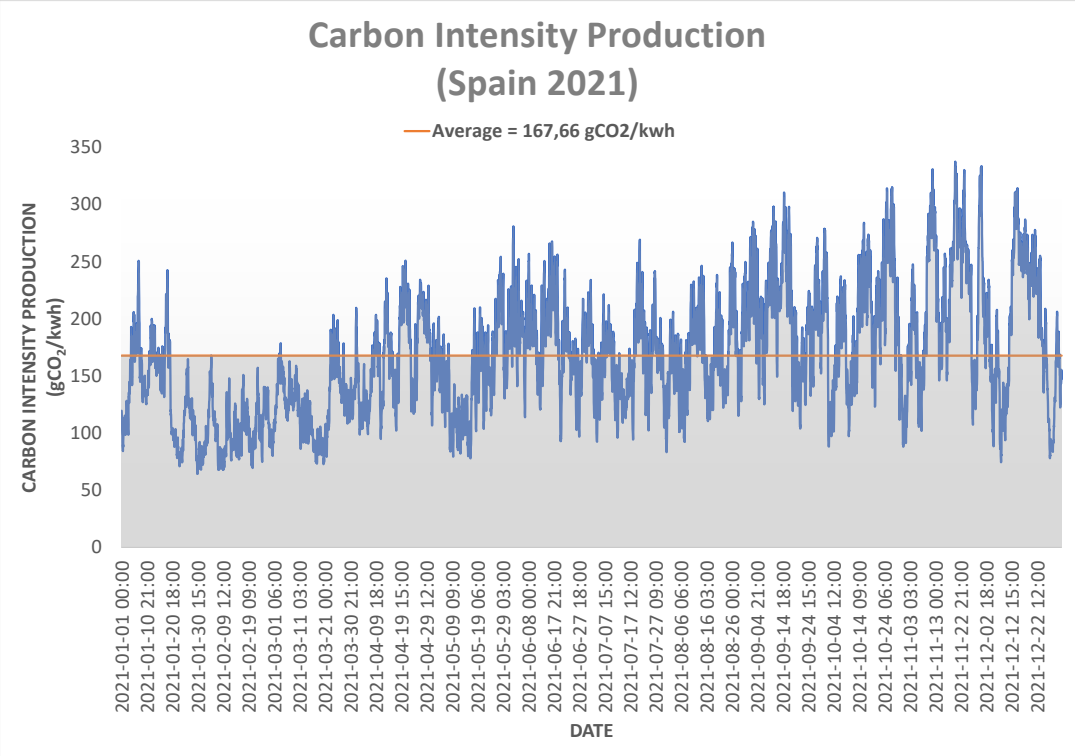


Figure 45: Carbon intensity production (Spain 2021). *Source:* [17]

Conclusions

The main project objective was to research about Body Gestures recognition for Human Robot Interaction and propose a solution for gesture classification using Deep Learning models. The submitted model has been based on a State-of-the-Art solution, which has an accuracy above any other alternatives ($> 90\%$).

Specifically, the proposed model uses the main key concepts of the State-of-the-Art solution with a few modifications that have allowed using a custom body gestures dataset.

As a feature extractor, the use of Media Pipe Pose has shown amazing results detecting joints and extracting landmarks, unfortunately, the precision of this method decreases when used against hands, and has some pending improvements. The fact that the extractor has only four joints in each hand and they do not correspond with the fingers, has been a bit problematic on gestures that require differentiating finger movements like *yes*.

A few tests have shown that gestures recognition applied to static gestures behaves quite better than on dynamic ones. So we understand, that for continuous gestures, the time component implies quite a challenge.

As already said, the solution or model is based on, has astonishing results, so we could say that our main problem is on our data. (*Different gestures length, camera configurations, resolution...*)

Despite all of this, and thanks to all the experiments done, we have been able to increase our accuracy with all the gestures, up to 60%.

Furthermore, POSTURE has been trained and tested with real-life situations with continuous gestures by using a batch approach that helps with the gestures classification.

Independently of the achieved results, is interesting to notice that all this research has paid off since a first paper was landed in RO-MAN and a second one is planned for Humanoids. [24]

Future Work

Although the dataset has been one of the main key factor of the model, it is also one of the biggest problems it has. One of the next steps to follow could be, first test our model with NTU dataset and see how it behaves and afterwards, transfer the learning.

In Deep Learning exists the transfer learning concept in which, for example, a model is trained with really good data and then all this neural network weights are stored and used against the new data that ends up being used to polish the model.

With this, we would expect the NN has been able to understand the gestures and apply all it has previously learned from our data.

Lessons Learned

While working on the project, a few interesting lessons learned topics appeared. We will not include all of them here, but only the ones that well-deserve it.

- Despite what is specified on the frameworks (PyG & PyGT) instructions, do not use a version of PyTorch newer than 1.8.1.

```
conda install pytorch==1.8.1
              torchvision==0.9.1
              torchaudio==0.8.1
              cpuonly -c pytorch

conda install pyg -c pyg -c conda-forge
pip install torch-geometric-temporal
```

- Decide from the beginning if you are going to use a CPU or GPU, otherwise you can reach the undesired situation on which you need to swap from one to the other and have half of your code that is only ready for CPU but with a GPU installation. (*Numpy is CPU only!*)
- Do not hesitate to use visualization tooling like Tensorboard, they will be really helpful.
- Organize your experiments with clear names and do not only store accuracy, but random images from test data, confusion matrices, etc.

Bibliography

[1] *Agile vs Waterfall: Difference Between Two Powerful Methodologies*. URL: https://hygger.io/guides/agile/agile-vs-waterfall/.

[2] Michael Argyle. *Non-verbal communication in human social interaction*. Cambridge University Press, 1972.

[3] René De La Barré et al. "Touchless interaction-novel chances and challenges". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 5611 LNCS (PART 2 2009), pp. 161–169. ISSN: 03029743. DOI: 10.1007/978-3-642-02577-8_18/COVER/. URL: https://link.springer.com/chapter/10.1007/978-3-642-02577-8_18.

[4] Christoph Bartneck et al. "Human-Robot Interaction: An Introduction". In: (Feb. 2020). DOI: 10.1017/9781108676649. URL: https://www.cambridge.org/core/books/humanrobot-interaction/2C042DEB4D0ECFFA5485857314E885BC.

[5] *Batch Normalization Definition*. URL: https://deepai.org/machine-learning-glossary-and-terms/batch-normalization.

[6] Gibran Benitez-Garcia et al. "IPN Hand: A video dataset and benchmark for real-time continuous hand gesture recognition". In: *Proceedings - International Conference on Pattern Recognition* (2020), pp. 4340–4347. ISSN: 10514651. DOI: 10.1109/ICPR48806.2021.9412317.

[7] Shaked Brody, Uri Alon, and Eran Yahav. "How Attentive are Graph Attention Networks?" In: *ICLR 2022* (May 2021). URL: https://arxiv.org/abs/2105.14491v2.

[8] *BUDDY PRO - Your Robot User Interaction Solutions for Your Brand Image*. URL: https://buddytherobot.com/en/buddy-pro/.

[9] Amar Budhiraja. *Dropout in (Deep) Machine learning*. Dec. 2016. URL: https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5.

[10] Tathagata Chakraborti et al. "AI Challenges in Human-Robot Cognitive Teaming". In: (July 2017). DOI: 10.48550/arxiv.1707.04775. URL: https://arxiv.org/abs/1707.04775v2.

[11] Bo Chen et al. "Online control programming algorithm for human–robot interaction system with a novel real-time human gesture recognition method:" in: *https://doi.org/10.1177/1729881419861764* 16 (4 July 2019). ISSN: 17298814. DOI: 10.1177/1729881419861764. URL: https://journals.sagepub.com/doi/full/10.1177/1729881419861764.

[12] Ke Cheng et al. "Skeleton-based action recognition with shift graph convolutional network". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2020), pp. 180–189. ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.00026.

[13] Cruzcampo. *Así se hizo ConMuchoAcento*. Jan. 2021. URL: https://www.youtube.com/watch?v=BQLTRMYHwvE.

[14] *DALL·E 2*. URL: https://openai.com/dall-e-2/.

[15] DeepFindr. *Understanding Graph Attention Networks*. Apr. 2021. URL: https://www.youtube.com/watch?v=A-yKQamf2Fc.

[16] DeepFindr. *Understanding Graph Neural Networks*. Sept. 2020. URL: https://www.youtube.com/watch?v=fOctJB4kVlM.

[17] *electricityMap – The leading resource for 24/7 CO2 grid data worldwide*. URL: https://electricitymap.org/.

[18] Harshala Gammulle et al. "Multi-modal Fusion for Single-Stage Continuous Gesture Recognition". In: *IEEE Transactions on Image Processing* 30 (Nov. 2020), pp. 7689–7701. DOI: 10.1109/TIP.2021.3108349. URL: http://arxiv.org/abs/2011.04945; http://dx.doi.org/10.1109/TIP.2021.3108349.

[19] Qing Gao et al. "Static hand gesture recognition with parallel CNNs for space human-robot interaction". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 10462 LNAI (2017), pp. 462–473. ISSN: 16113349. DOI: 10.1007/978-3-319-65289-4_44/TABLES/3. URL: https://link.springer.com/chapter/10.1007/978-3-319-65289-4_44.

[20] A. Garrell et al. "Proactive behavior of an autonomous mobile robot for human-assisted learning". In: *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication* (2013), pp. 107–113. DOI: 10.1109/ROMAN.2013.6628463.

[21] Beatrice De Gelder. "Why bodies? Twelve reasons for including bodily expressions in affective neuroscience". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 364 (1535 Dec. 2009), p. 3475. ISSN: 14712970. DOI: 10.1098/RSTB.2009.0190. URL: /pmc/articles/PMC2781896/; /pmc/articles/PMC2781896/?report=abstract; https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2781896/.

[22] *Google Colab*. URL: https://colab.research.google.com/.

[23] Robert A. Hinde. *Non-verbal communication*. Cambridge University Press, Nov. 1972, p. 443. ISBN: 978-0521290128.

[24] *Humanoids 2022 - IEEE Robotics and Automation Society*. URL: https://www.ieee-ras.org/conferences-workshops/fully-sponsored/humanoids; https://www.humanoids2022.org/.

[25] *IRI - Institut de Robòtica i Informàtica industrial*. URL: https://www.iri.upc.edu/.

[26] Shruti Jadon. *Introduction to Different Activation Functions for Deep Learning*. 2018. URL: https://medium.com/@shrutijadon/survey-on-activation-functions-for-deep-learning-9689331ba092.

[27] *joanjaumeoliver/GESTURE-PROJECT*. URL: https://github.com/joanjaumeoliver/GESTURE-PROJECT.

[28] *Keras: the Python deep learning API*. URL: https://keras.io/.

[29] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings* (Sept. 2016). URL: https://arxiv.org/abs/1609.02907v4.

[30] Ria Kulshrestha. *Understanding Attention In Deep Learning (NLP)*. May 2020. URL: https://towardsdatascience.com/attaining-attention-in-deep-learning-a712f93bdb1e.

[31] Huang Kung-Hsiang. *A Gentle Introduction to Graph Neural Networks (Basics, DeepWalk, and GraphSage)*. Feb. 2019. URL: https://towardsdatascience.com/a-gentle-introduction-to-graph-neural-network-basics-deepwalk-and-graphsage-db5d540d50b3.

[32] Jun Kwan, Chinkye Tan, and Akansel Cosgun. "Gesture Recognition for Initiating Human-to-Robot Handovers". In: (July 2020). DOI: 10.48550/arxiv.2007.09945. URL: https://arxiv.org/abs/2007.09945v2.

[33] Javier Laplaza et al. "Body Gesture Recognition to Control a Social Robot". In: (June 2022). DOI: 10.48550/arxiv.2206.07538. URL: https://arxiv.org/abs/2206.07538v1.

[34] Maosen Li et al. "Actional-structural graph convolutional networks for skeleton-based action recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2019-June (June 2019), pp. 3590–3598. ISSN: 10636919. DOI: 10.1109/CVPR.2019.00371.

[35] Jun Liu et al. "NTU RGB+D 120: A Large-Scale Benchmark for 3D Human Activity Understanding". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42 (10 May 2019), pp. 2684–2701. DOI: 10.1109/TPAMI.2019.2916873. URL: http://arxiv.org/abs/1905.04757; http://dx.doi.org/10.1109/TPAMI.2019.2916873.

[36] Ziyu Liu et al. "Disentangling and unifying graph convolutions for skeleton-based action recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (2020), pp. 140–149. ISSN: 10636919. DOI: 10.1109/CVPR42600.2020.00022.

[37] Antonio Longa. *Pytorch Geometric tutorial: Graph attention networks (GAT) implementation*. Mar. 2021. URL: https://www.youtube.com/watch?v=CwsPoa7z2c8.

[38] *LuxAI - Award winning social robots for autism and special needs education*. URL: https://luxai.com/.

[39] *Master's Degree in Industrial Engineering Barcelona School of Industrial Engineering (ETSEIB) — UPC. Universitat Politècnica de Catalunya*. URL: https://muei.etseib.masters.upc.edu/en.

[40] Mediapipe. *Holistic*. URL: https://google.github.io/mediapipe/solutions/holistic.html.

[41] Mediapipe. *Pose*. URL: https://google.github.io/mediapipe/solutions/pose.

[42] Sepehr Mohaimenianpour and Richard Vaughan. "Hands and Faces, Fast: Mono-Camera User Detection Robust Enough to Directly Control a UAV in Flight". In: *IEEE International Conference on Intelligent Robots and Systems* (Dec. 2018), pp. 5224–5231. ISSN: 21530866. DOI: 10.1109/IROS.2018.8593709.

[43] Andrew Ng. *Deep Learning*. URL: https://www.coursera.org/specializations/deep-learning.

[44] Kai Nickel and Rainer Stiefelhagen. "Visual recognition of pointing gestures for human–robot interaction". In: *Image and Vision Computing* 25 (12 Dec. 2007), pp. 1875–1884. ISSN: 0262-8856. DOI: 10.1016/J.IMAVIS.2005.12.020.

[45] Fatemeh Noroozi et al. "Survey on Emotional Body Gesture Recognition". In: *IEEE Transactions on Affective Computing* 12 (2 Apr. 2021), pp. 505–523. ISSN: 19493045. DOI: 10.1109/TAFFC.2018.2874986.

[46] Tim Tim O'Sullivan et al. *Key concepts in communication and cultural studies | Jey Tim*. 1994. URL: https://www.academia.edu/1999782/Key_concepts_in_communication_and_cultural_studies.

[47] Joan Jaume Oliver. *joanjaumeoliver/IRIGesture*. URL: https://github.com/joanjaumeoliver/IRIGesture.

[48] *PyCharm: the Python IDE for Professional Developers by JetBrains*. URL: https://www.jetbrains.com/pycharm/.

[49] *PyTorch*. URL: https://pytorch.org/.

[50] *PyTorch Geometric*. URL: https://pytorch-geometric.readthedocs.io/en/latest/.

[51] *PyTorch Geometric Temporal*. URL: https://pytorch-geometric-temporal.readthedocs.io/en/latest/.

[52] Jagdish Lal Raheja, Mona Chandra, and Ankit Chaudhary. "3D gesture based real-time object selection and recognition". In: *Pattern Recognition Letters* 115 (Nov. 2018), pp. 14–19. ISSN: 0167-8655. DOI: 10.1016/J.PATREC.2017.09.034.

[53] *RO-MAN 2022 - IEEE Robotics and Automation Society*. URL: https://www.ieee-ras.org/about-ras/ras-calendar/event/2107-ro-man-2022; http://www.ro-man2022.org/.

[54] Ramón Romero. *Study of deep learning techniques for human gesture classification*. Sept. 2021. URL: https://upcommons.upc.edu/handle/2117/352487.

[55] Ramon Romero. *GESTURE DATASET*. URL: https://github.com/RamonRL/GESTURE-PROJECT/tree/main/dataset/BodyGestureDataset.

[56] Benedek Rozemberczki et al. "PyTorch Geometric Temporal: Spatiotemporal Signal Processing with Neural Machine Learning Models". In: *International Conference on Information and Knowledge Management, Proceedings* (Apr. 2021), pp. 4564–4573. DOI: 10.48550/arxiv.2104.07788. URL: https://arxiv.org/abs/2104.07788v3.

[57] Rabeya Tus Sadia. *Understanding Graph Neural Network with hands-on example*. July 2020. URL: https://medium.com/@rtsrumi07/understanding-graph-neural-network-with-hands-on-example-part-1-6e35d7fe2777; https://medium.com/@rtsrumi07/understanding-graph-neural-network-with-hands-on-example-part-2-139a691ebeac.

[58] Behzad Sadrfaridpour and Yue Wang. "Collaborative Assembly in Hybrid Manufacturing Cells: An Integrated Framework for Human-Robot Interaction". In: *IEEE Transactions on Automation Science and Engineering* 15 (3 July 2018), pp. 1178–1192. ISSN: 15455955. DOI: 10.1109/TASE.2017.2748386.

[59] Franco Scarselli et al. "The graph neural network model". In: *IEEE Transactions on Neural Networks* 20 (1 Jan. 2009), pp. 61–80. ISSN: 10459227. DOI: 10.1109/TNN.2008.2005605.

[60] Amir Shahroudy et al. "NTU RGB+D: A Large Scale Dataset for 3D Human Activity Analysis". In: *Computer Vision Foundation* (2016).

[61] Abdel-Nasser Sharkawy. "Human-Robot Interaction: Applications". In: *arXiv preprint* (Feb. 2021), pp. 3–5. DOI: 10.48550/arxiv.2102.00928. URL: https://arxiv.org/abs/2102.00928v1.

[62] Lei Shi et al. "Decoupled Spatial-Temporal Attention Network for Skeleton-Based Action Recognition". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12626 LNCS (July 2020), pp. 38–53. ISSN: 16113349. DOI: 10.48550/arxiv.2007.03263. URL: https://arxiv.org/abs/2007.03263v1.

[63] Lei Shi et al. "Skeleton-based action recognition with directed graph neural networks". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2019-June (June 2019), pp. 7904–7913. ISSN: 10636919. DOI: 10.1109/CVPR.2019.00810.

[64] Lei Shi et al. "Two-Stream Adaptive Graph Convolutional Networks for Skeleton-Based Action Recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* 2019-June (May 2018), pp. 12018–12027. ISSN: 10636919. DOI: 10.48550/arxiv.1805.07694. URL: https://arxiv.org/abs/1805.07694v3.

[65] Seong Og Shin, Donghan Kim, and Yong Ho Seo. "Controlling mobile robot using IMU and EMG sensor-based gesture recognition". In: *Proceedings - 2014 9th International Conference on Broadband and Wireless Computing, Communication and Applications, BWCCA 2014* (Jan. 2014), pp. 554–557. DOI: 10.1109/BWCCA.2014.145.

[66] Yi Fan Song et al. "Richly Activated Graph Convolutional Network for Robust Skeleton-Based Action Recognition". In: *IEEE Transactions on Circuits and Systems for Video Technology* 31 (5 May 2021), pp. 1915–1925. ISSN: 15582205. DOI: 10.1109/TCSVT.2020.3015051.

[67] *TensorFlow*. URL: https://www.tensorflow.org/.

[68] Kalpit Thakkar and P. J. Narayanan. "Part-based Graph Convolutional Network for Action Recognition". In: *British Machine Vision Conference 2018, BMVC 2018* (Sept. 2018). DOI: 10.48550/arxiv.1809.04983. URL: https://arxiv.org/abs/1809.04983v1.

[69] Ming Fong Tsai and Chiung Hung Chen. "Spatial Temporal Variation Graph Convolutional Networks (STV-GCN) for Skeleton-Based Emotional Action Recognition". In: *IEEE Access* 9 (2021), pp. 13870–13877. ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3052246.

[70] Ashish Vaswani et al. "Attention Is All You Need". In: *Advances in Neural Information Processing Systems* 2017-December (June 2017), pp. 5999–6009. ISSN: 10495258. URL: https://arxiv.org/abs/1706.03762v5.

[71] Carlos Santana Vega. *¿Por qué estas redes neuronales son tan potentes?* Nov. 2021. URL: https://www.youtube.com/watch?v=xi94v_jl26U.

[72] Carlos Santana Vega. *Las redes neuronales ahora prestan atención!* Sept. 2021. URL: https://www.youtube.com/watch?v=aL-EmKuB078.

[73] Petar Veličković et al. "Graph Attention Networks". In: *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (Oct. 2017). URL: https://arxiv.org/abs/1710.10903v3.

[74] Tom Williams et al. "Virtual, Augmented, and Mixed Reality for Human-Robot Interaction (VAM-HRI)". In: *ACM/IEEE International Conference on Human-Robot Interaction* 2019-March (Mar. 2019), pp. 671–672. ISSN: 21672148. DOI: 10.1109/HRI.2019.8673207.

[75] Pradyumna Yadav. *The journey of Gradient Descent — From Local to Global*. Feb. 2021. URL: https://medium.com/analytics-vidhya/journey-of-gradient-descent-from-local-to-global-c851eba3d367.

[76] Sijie Yan, Yuanjun Xiong, and Dahua Lin. "Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018* (Jan. 2018), pp. 7444–7452. DOI: 10.48550/arxiv.1801.07455. URL: https://arxiv.org/abs/1801.07455v2.

[77] Hao Yang et al. "Feedback Graph Convolutional Network for Skeleton-Based Action Recognition". In: *IEEE Transactions on Image Processing* 31 (2022), pp. 164–175. ISSN: 19410042. DOI: 10.1109/TIP.2021.3129117.

[78] Jie Zhou et al. "Graph Neural Networks: A Review of Methods and Applications". In: *AI Open* 1 (Dec. 2018), pp. 57–81. ISSN: 26666510. DOI: 10.48550/arxiv.1812.08434. URL: https://arxiv.org/abs/1812.08434v4;https://arxiv.org/abs/1812.08434v6.

Annex A - Graphs

A1 *GNN - Different variants of recurrent operators*

Name	Variant	Aggregator	Updater
Spectral Methods	ChebNet	$\mathbf{N}_k = \mathbf{T}_k(\tilde{\mathbf{L}})\mathbf{X}$	$\mathbf{H} = \sum_{k=0}^{K} \mathbf{N}_k \mathbf{\Theta}_k$
	1^{st}-order model	$\mathbf{N}_0 = \mathbf{X}$ $\mathbf{N}_1 = \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}}\mathbf{X}$	$\mathbf{H} = \mathbf{N}_0\mathbf{\Theta}_0 + \mathbf{N}_1\mathbf{\Theta}_1$
	Single parameter	$\mathbf{N} = (\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})\mathbf{X}$	$\mathbf{H} = \mathbf{N}\mathbf{\Theta}$
	GCN	$\mathbf{N} = \tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}$	$\mathbf{H} = \mathbf{N}\mathbf{\Theta}$
Non-spectral Methods	Neural FPs	$\mathbf{h}^t_{\mathcal{N}_v} = \mathbf{h}^{t-1}_v + \sum_{k=1}^{\mathcal{N}_v} \mathbf{h}^{t-1}_k$	$\mathbf{h}^t_v = \sigma(\mathbf{h}^t_{\mathcal{N}_v}\mathbf{W}^{\mathcal{N}_v}_L)$
	DCNN	Node classification: $\mathbf{N} = \mathbf{P}^*\mathbf{X}$ Graph classification: $\mathbf{N} = 1^T_N\mathbf{P}^*\mathbf{X}/N$	$\mathbf{H} = f\left(\mathbf{W}^c \odot \mathbf{N}\right)$
	GraphSAGE	$\mathbf{h}^t_{\mathcal{N}_v} = \text{AGGREGATE}_t\left(\{\mathbf{h}^{t-1}_u, \forall u \in \mathcal{N}_v\}\right)$	$\mathbf{h}^t_v = \sigma\left(\mathbf{W}^t \cdot [\mathbf{h}^{t-1}_v \| \mathbf{h}^t_{\mathcal{N}_v}]\right)$
Graph Attention Networks	GAT	$\alpha_{vk} = \frac{\exp\left(\text{LeakyReLU}\left(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_k]\right)\right)}{\sum_{j\in\mathcal{N}_v}\exp(\text{LeakyReLU}(\mathbf{a}^T[\mathbf{W}\mathbf{h}_v \| \mathbf{W}\mathbf{h}_j]))}$ $\mathbf{h}^t_{\mathcal{N}_v} = \sigma\left(\sum_{k\in\mathcal{N}_v}\alpha_{vk}\mathbf{W}\mathbf{h}_k\right)$ Multi-head concatenation: $\mathbf{h}^t_{\mathcal{N}_v} = \Big\|_{m=1}^{M}\sigma\left(\sum_{k\in\mathcal{N}_v}\alpha^m_{vk}\mathbf{W}^m\mathbf{h}_k\right)$ Multi-head average: $\mathbf{h}^t_{\mathcal{N}_v} = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{k\in\mathcal{N}_v}\alpha^m_{vk}\mathbf{W}^m\mathbf{h}_k\right)$	$\mathbf{h}^t_v = \mathbf{h}^t_{\mathcal{N}_v}$
Gated Graph Neural Networks	GGNN	$\mathbf{h}^t_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{h}^{t-1}_k + \mathbf{b}$	$\mathbf{z}^t_v = \sigma(\mathbf{W}^z\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^z\mathbf{h}^{t-1}_v)$ $\mathbf{r}^t_v = \sigma(\mathbf{W}^r\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}^r\mathbf{h}^{t-1}_v)$ $\widetilde{\mathbf{h}^t_v} = \tanh(\mathbf{W}\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{U}(\mathbf{r}^t_v \odot \mathbf{h}^{t-1}_v))$ $\mathbf{h}^t_v = (1 - \mathbf{z}^t_v) \odot \mathbf{h}^{t-1}_v + \mathbf{z}^t_v \odot \widetilde{\mathbf{h}^t_v}$
Graph LSTM	Tree LSTM (Child sum)	$\mathbf{h}^t_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{h}^{t-1}_k$	$\mathbf{i}^t_v = \sigma(\mathbf{W}^i\mathbf{x}^t_v + \mathbf{U}^i\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{b}^i)$ $\mathbf{f}^t_{vk} = \sigma\left(\mathbf{W}^f\mathbf{x}^t_v + \mathbf{U}^f\mathbf{h}^{t-1}_k + \mathbf{b}^f\right)$ $\mathbf{o}^t_v = \sigma(\mathbf{W}^o\mathbf{x}^t_v + \mathbf{U}^o\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{b}^o)$ $\mathbf{u}^t_v = \tanh(\mathbf{W}^u\mathbf{x}^t_v + \mathbf{U}^u\mathbf{h}^t_{\mathcal{N}_v} + \mathbf{b}^u)$ $\mathbf{c}^t_v = \mathbf{i}^t_v \odot \mathbf{u}^t_v + \sum_{k\in\mathcal{N}_v}\mathbf{f}^t_{vk} \odot \mathbf{c}^{t-1}_k$ $\mathbf{h}^t_v = \mathbf{o}^t_v \odot \tanh(\mathbf{c}^t_v)$
	Tree LSTM (N-ary)	$\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{l=1}^{K}\mathbf{U}^i_l\mathbf{h}^{t-1}_{vl}$ $\mathbf{h}^{tf}_{\mathcal{N}_v k} = \sum_{l=1}^{K}\mathbf{U}^f_{kl}\mathbf{h}^{t-1}_{vl}$ $\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{l=1}^{K}\mathbf{U}^o_l\mathbf{h}^{t-1}_{vl}$ $\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{l=1}^{K}\mathbf{U}^u_l\mathbf{h}^{t-1}_{vl}$	$\mathbf{i}^t_v = \sigma(\mathbf{W}^i\mathbf{x}^t_v + \mathbf{h}^{ti}_{\mathcal{N}_v} + \mathbf{b}^i)$ $\mathbf{f}^t_{vk} = \sigma(\mathbf{W}^f\mathbf{x}^t_v + \mathbf{h}^{tf}_{\mathcal{N}_v k} + \mathbf{b}^f)$ $\mathbf{o}^t_v = \sigma(\mathbf{W}^o\mathbf{x}^t_v + \mathbf{h}^{to}_{\mathcal{N}_v} + \mathbf{b}^o)$ $\mathbf{u}^t_v = \tanh(\mathbf{W}^u\mathbf{x}^t_v + \mathbf{h}^{tu}_{\mathcal{N}_v} + \mathbf{b}^u)$ $\mathbf{c}^t_v = \mathbf{i}^t_v \odot \mathbf{u}^t_v + \sum_{l=1}^{K}\mathbf{f}^t_{vl} \odot \mathbf{c}^{t-1}_{vl}$ $\mathbf{h}^t_v = \mathbf{o}^t_v \odot \tanh(\mathbf{c}^t_v)$
	Graph LSTM	$\mathbf{h}^{ti}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^i_{m(v,k)}\mathbf{h}^{t-1}_k$ $\mathbf{h}^{to}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^o_{m(v,k)}\mathbf{h}^{t-1}_k$ $\mathbf{h}^{tu}_{\mathcal{N}_v} = \sum_{k\in\mathcal{N}_v}\mathbf{U}^u_{m(v,k)}\mathbf{h}^{t-1}_k$	$\mathbf{i}^t_v = \sigma(\mathbf{W}^i\mathbf{x}^t_v + \mathbf{h}^{ti}_{\mathcal{N}_v} + \mathbf{b}^i)$ $\mathbf{f}^t_{vk} = \sigma(\mathbf{W}^f\mathbf{x}^t_v + \mathbf{U}^f_{m(v,k)}\mathbf{h}^{t-1}_k + \mathbf{b}^f)$ $\mathbf{o}^t_v = \sigma(\mathbf{W}^o\mathbf{x}^t_v + \mathbf{h}^{to}_{\mathcal{N}_v} + \mathbf{b}^o)$ $\mathbf{u}^t_v = \tanh(\mathbf{W}^u\mathbf{x}^t_v + \mathbf{h}^{tu}_{\mathcal{N}_v} + \mathbf{b}^u)$ $\mathbf{c}^t_v = \mathbf{i}^t_v \odot \mathbf{u}^t_v + \sum_{k\in\mathcal{N}_v}\mathbf{f}^t_{vk} \odot \mathbf{c}^{t-1}_k$ $\mathbf{h}^t_v = \mathbf{o}^t_v \odot \tanh(\mathbf{c}^t_v)$

Table 11: Different variants of recurrent operators. *Source* [*78*]

A2 *TGNN - Models*

Model	Temporal Layer	GNN Layer
DCRNN	GRU	DiffConv
GConvGRU	GRU	Chebyshev
GConvLSTM	LSTM	Chebyshev
GC-LSTM	LSTM	Chebyshev
DyGrAE	LSTM	GGCN
LRGCN	LSTM	RGCN
EGCN-H	GRU	GCN
EGCN-O	LSTM	GCN
T-GCN	GRU	GCN
A3T-GCN	GRU	GCN
AGCRN	GRU	Chebyshev
MPNN LSTM	LSTM	GCN
STGCN	Attention	Chebyshev
ASTGCN	Attention	Chebyshev
MSTGCN	Attention	Chebyshev
GMAN	Attention	Custom
MTGNN	Attention	Custom
AAGCN	Attention	Custom

Table 12: Common Spatiotemporal deep learning models based on the temporal and spatial block. *Source:* [56]

Annex B - Code

All the code developed for this thesis can be found below or in its GitHub repository [47]. Mainly, it has been organized in four different files:

- **Model**: The classifier object with all the layers.
- **Main**: The Main file which calls and uses all the others.
- **Dataset**: It's the parser, which downloads data from GitHub and parses as needed.
- **Tools**: Contains generic functions that have been used over the project.

B1 *Model.py*

```python
import torch
import torch.nn as nn
from torch_geometric_temporal import AAGCN


class Classifier(nn.Module):

    def __init__(
            self,
            edge_index: torch.LongTensor,
            in_channels: int = 4,
            out_channels: int = 8,
            num_nodes: int = 15,
            num_subsets: int = 15,
            device: str = 'cpu',
    ):
        super(Classifier, self).__init__()
        self.initialAAGCN = AAGCN(in_channels, 64, edge_index, num_nodes,
                                  stride=1, residual=True,
                                  adaptive=True, attention=True, device=device)

        self.middleAAGCN = nn.ModuleList()
        for i in range(num_subsets):
            self.middleAAGCN.append(AAGCN(64, 64, edge_index, num_nodes,
                                          stride=1, residual=True,
                                          adaptive=True, attention=True))

        self.finalAAGCN = AAGCN(64, out_channels, edge_index,
                                num_nodes, stride=1, residual=True,
                                adaptive=True, attention=True)

        self.linear = torch.nn.Linear(30 * 15, out_channels)

    def forward(self, x):
        # x = [batch_size, features_per_node, frames, number_nodes]
        y = self.initialAAGCN(x)  # y = [64, 64, 30, 15]

        for i in range(len(self.middleAAGCN)):
            y = self.middleAAGCN[i](y)  # y = [64, 64, 30, 15]

```

```
41          y = self.finalAAGCN(y)  # y = [64, 8, 30, 15]
42          # Test Stride = 1 in final AAGCN
43
44          values, _ = torch.max(y.data, 1)  # values = [64, 30, 15]
45          flatten_values = torch.flatten(values, 1)
46          y_pred = self.linear(flatten_values)
47          return y_pred
```

B2 *Main.py*

```
import os
import random
import typing
from pathlib import Path

import sys

import torch
import torch.nn.functional
from torch.utils.tensorboard import SummaryWriter

import utils.tools as tools
from dataset.IRIDatasetTemporal import IRIGestureTemporal
from model.AAGCN import Classifier


def train(categories: typing.List[str], tensorboard_name: str):
    step = 0
    loss_list = []
    acc_list = []
    total_guesses = torch.zeros(0).to(DEVICE)
    total_labels = torch.zeros(0).to(DEVICE)
    for encoder_inputs, labels, paths_idx in train_loader:
        total_labels = torch.cat((total_labels, labels))

        optimizer.zero_grad()
        y_hat = model(encoder_inputs)  # Get model predictions
        loss = loss_fn(y_hat.float(), labels.long())
        loss.backward()

        optimizer.step()
        step = step + 1
        loss_list.append(loss.item())

        # Softmax is implicit in Loss but not in Acc
        y_hat_softmax = torch.softmax(y_hat, dim=1)
        guessed_list = torch.argmax(y_hat_softmax, dim=1)
        total_guesses = torch.cat((total_guesses, guessed_list))
        corrects = torch.flatten((guessed_list == labels).float())
        acc = corrects.sum() / len(corrects)
        acc_list.append(acc.item())

        if step \% 5 == 0:
            print("Loss = " + str(sum(loss_list) / len(loss_list)))
            print("Acc = " + str(sum(acc_list) / len(acc_list)))
    scheduler.step()
    print("Epoch {} train CrossEntropyLoss:"
          " {:.4f} Acc: {:.4f}".format(epoch,
                                       sum(loss_list) / len(loss_list),
                                       sum(acc_list) / len(acc_list)))

    if epoch \% 100 == 0:
```

```
53          writer.add_figure("TrainConfusionMatrix",
54                            tools.create_confusion_matrix(
55                                total_guesses, total_labels, categories,
56                                f'Train-Epoch:{epoch}'), epoch)
57      writer.add_scalar('Loss/Train', sum(loss_list) / len(loss_list), epoch)
58      writer.add_scalar('Accuracy/Train', sum(acc_list) / len(acc_list), epoch)
59
60      if tensorboard_adv_mode:
61          for idx, p in enumerate(model.parameters()):
62              if p.grad is not None:
63                  writer.add_scalar(f'TrainGradients/grad_{idx}',
64                                    p.grad.norm(), epoch)
65
66          writer.add_hparams({'lr': scheduler.get_last_lr()[0]},
67                             {'accuracy': sum(acc_list) / len(acc_list),
68                              'loss': sum(loss_list) / len(loss_list)})
69
70      if model_save:
71          torch.save(model.state_dict(),
72                     os.path.join(Path().absolute(),
73                                  'checkpoints',
74                                  f'{tensorboard_name}_Checkpoints',
75                                  'Epoch_' + str(epoch) + '.pth'))
76
77
78  def test(dataset_videos_paths: typing.List[str],
79           categories: typing.List[str]):
80      model.eval()
81      batch = 0
82      # Store for analysis
83      total_loss = []
84      total_acc = []
85      total_guesses = torch.zeros(0).to(DEVICE)
86      total_labels = torch.zeros(0).to(DEVICE)
87      for encoder_inputs, labels, paths_idx in test_loader:
88          # Get model predictions
89          total_labels = torch.cat((total_labels, labels))
90          y_hat = model(encoder_inputs)
91          # Mean squared error
92          loss = loss_fn(y_hat.float(), labels.long())
93          total_loss.append(loss.item())
94
95          y_hat_softmax = torch.softmax(y_hat, dim=1)
96          guessed_list = torch.argmax(y_hat_softmax, dim=1)
97          total_guesses = torch.cat((total_guesses, guessed_list))
98          corrects_list = (guessed_list == labels).float()
99          corrects = torch.flatten(corrects_list)
100         acc = corrects.sum() / len(corrects)
101         total_acc.append(acc.item())
102
103         if tensorboard_adv_mode:
104             video_idx = random.choice(paths_idx.tolist())
105             idx = (paths_idx == video_idx).nonzero(as_tuple=True)[0]
106             video_path = dataset_videos_paths[video_idx]
```

```
107             video_name = os.path.splitext(os.path.basename(video_path))[0]
108             video_label = categories[int(labels[idx])]
109             guessed_label = categories[int(guessed_list[idx])]
110 
111             writer.add_video(f'{video_label}/{video_name}',
112                              tools.read_video(video_path), batch)
113             writer.add_text(f'{video_label}/{video_name}',
114                             f'Guessed {guessed_label}', batch)
115 
116         batch += 1
117 
118     print('Test CrossEntropyLoss: {:.4f} '
119           'Acc: {:.4f}'.format(sum(total_loss) / len(total_loss),
120                               sum(total_acc) / len(total_acc)))
121 
122     if tensorboard_adv_mode:
123         writer.add_figure("TestConfusionMatrix",
124                           tools.create_confusion_matrix(
125                               total_guesses,
126                               total_labels, categories,
127                               f'Test-Epoch:{epoch}'), epoch)
128     writer.add_scalar('Loss/Test',
129                       sum(total_loss) / len(total_loss), epoch)
130     writer.add_scalar('Accuracy/Test',
131                       sum(total_acc) / len(total_acc), epoch)
132 
133 
134 tools.seed_everything()
135 
136 DEVICE = torch.device(sys.argv[1])
137 tensorboard_adv_mode = sys.argv[3]  # True False
138 model_save = sys.argv[4]  # True False
139 shuffle = True
140 batch_size = 128
141 
142 run_name = f'{input("Add TensorBoard RUN Name")}'
143 loader = IRIGestureTemporal(os.path.join(Path().absolute(),
144                                          'dataset'),
145                             dataTypes="All", token=sys.argv[2])
146 dataset = loader.get_all_dataset()
147 dataset.shuffle()
148 train_dataset, test_dataset = \
149     tools.temporal_dataset_split(dataset,
150                                  train_ratio=0.95)
151 print("Dataset type:  ", dataset)
152 print("Number of samples / sequences: ", len(set(dataset)))
153 
154 print("Number of train buckets: ", len(set(train_dataset)))
155 print("Number of test buckets: ", len(set(test_dataset)))
156 
157 # Creating Data loaders
158 train_loader = tools.create_data_loaders(train_dataset,
159                                          batch_size,
160                                          shuffle,
```

```
                                          DEVICE)
test_loader = tools.create_data_loaders(test_dataset,
                                        32, shuffle,
                                        DEVICE)

# Create model and optimize
model = Classifier(
    edge_index=train_dataset.get_static_edge_index().to(DEVICE),
    out_channels=len(loader.categories),
    device=DEVICE.type)
model.to(DEVICE)

optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
scheduler = torch.optim.lr_scheduler.ExponentialLR(optimizer, gamma=0.95)
loss_fn = torch.nn.CrossEntropyLoss(
    weight=dataset.get_classes_proportion_vector().float().to(DEVICE))

tools.clear_path(
    os.path.join(Path().absolute(), 'checkpoints',
                 f'{run_name}_Checkpoints'))
writer = SummaryWriter(log_dir=os.path.join('tensorboard/runs',
                                            run_name))

model.train()
epoch = 0
max_epochs = 3000
while True:
    if epoch < max_epochs:
        train(categories=loader.categories, tensorboard_name=run_name)
        epoch += 1
        if epoch \% 25 == 0:
            test(dataset_videos_paths=test_dataset.videos_paths,
                 categories=loader.categories)
            model.train()
    elif input("Do you want to exit?") == 'Yes':
        break
    else:
        max_epochs = int(input("Set new max number of epochs"))

writer.flush()
writer.close()
```

B3 *Dataset.py*

```python
import base64
import glob
import os
import re
import shutil
from typing import List, Union
from typing import Tuple

import cv2
import mediapipe as mp
import numpy as np
import torch
from github import Github
from github import GithubException
from torch import default_generator, randperm
from torch_geometric.data import (InMemoryDataset, Data)
from torch_geometric_temporal.signal import DynamicGraphTemporalSignal

import utils.tools as tools

Edge_Indices = List[Union[np.ndarray, None]]
Edge_Weights = List[Union[np.ndarray, None]]
Node_Features = List[Union[np.ndarray, None]]
Targets = List[Union[np.ndarray, None]]
Additional_Features = List[np.ndarray]
Videos_Path = List[Union[str, None]]


class CustomDynamicGraphTemporalSignal(DynamicGraphTemporalSignal):
    def __init__(self,
                 videos_path: Videos_Path,
                 edge_indices: Edge_Indices,
                 edge_weights: Edge_Weights,
                 features: Node_Features,
                 targets: Targets,
                 **kwargs: Additional_Features):
        self.videos_paths = videos_path
        super(CustomDynamicGraphTemporalSignal,
              self).__init__(edge_indices,
                             edge_weights,
                             features,
                             targets, **kwargs)

    def shuffle(self):
        indices = randperm(self.snapshot_count,
                           generator=default_generator).tolist()

        self.videos_paths = tools.sort_list_by_indices(self.videos_paths,
                                                       indices)
        self.edge_indices = tools.sort_list_by_indices(self.edge_indices,
                                                       indices)
        self.edge_weights = tools.sort_list_by_indices(self.edge_weights,
```

```
53                                                          indices)
54          self.features = tools.sort_list_by_indices(self.features,
55                                                     indices)
56          self.targets = tools.sort_list_by_indices(self.targets,
57                                                    indices)
58
59      def get_static_edge_index(self):
60          return self.__get_item__(0).edge_index
61
62      def get_classes_proportion_vector(self):
63          proportion_vector = torch.zeros(len(self.targets[0]))
64          for e in self.targets:
65              proportion_vector += e
66          return proportion_vector
67
68
69  class IRIGestureTemporal(InMemoryDataset):
70      r"""The IRIGesture dataset
71      .. note::
72          TO DO
73      Args:
74          root (string): Root directory where the dataset should be saved.
75              would be returned.
76          testSubject (string): Subject to exclude from Dataset in order
77          to use as Test.
78          dataTypes (string): Use 'Dynamic' or 'Static'.
79          token (string, optional): GitHub token needed in order to download
80              IRIGesture dataset. (By default uses 'GITHUB_TOKEN' environment
81              variable)
82          categories (list, optional): List of categories to include in the
83              dataset. Can include the categories :obj:`"attention"`,
84              :obj:`"right"`, :obj:`"left"`, :obj:`"stop"`, :obj:`"yes"`,
85              :obj:`"shrug"`. If set to :obj:`None`, the dataset will
86              contain all categories. (default: :obj:`None`)
87          transform (callable, optional): A function/transform that takes in an
88              :obj:`torch_geometric.data.Data` object and returns a transformed
89              version. The data object will be transformed before every access.
90              (default: :obj:`None`)
91          pre_transform (callable, optional): A function/transform that takes in
92              an :obj:`torch_geometric.data.Data` object and returns a
93              transformed version. The data object will be transformed before
94              being saved to disk. (default: :obj:`None`)
95          pre_filter (callable, optional): A function that takes in an
96              :obj:`torch_geometric.data.Data` object and returns a boolean
97              value, indicating whether the data object should be included in
98               thefinal dataset. (default: :obj:`None`)
99      """
100     # We will use this information in order to access GitHub.
101     __owner = "RamonRL"
102     __repo = "GESTURE-PROJECT"
103     __serverPath = "dataset/BodyGestureDataset"
104
105     __categories = []
106     __categoriesStatic = ['attention', 'right', 'left',
```

```
107                          'stop', 'yes', 'shrug']  # , 'static', 'random']
108     __categoriesDynamic = ['greeting', 'continue',
109                          'turnback', 'no', 'slowdown', 'come']  # , 'back']
110
111     __nodes_to_use = [0,  # nose
112                       # 1,       # left_eye_inner
113                       # 2,       # left_eye
114                       # 3,       # left_eye_outer
115                       # 4,       # right_eye_inner
116                       # 5,       # right_eye
117                       # 6,       # right_eye_outer
118                       # 7,       # left_ear
119                       # 8,       # right_ear
120                       # 9,       # mouth_left
121                       # 10,      # mouth_right
122                       11,  # left_shoulder
123                       12,  # right_shoulder
124                       13,  # left_elbow
125                       14,  # right_elbow
126                       15,  # left_wrist
127                       16,  # right_wrist
128                       17,  # left_pinky
129                       18,  # right_pinky
130                       19,  # left_index
131                       20,  # right_index
132                       21,  # left_thumb
133                       22,  # right_thumb
134                       23,  # left_hip
135                       24  # right_hip
136                       # 25,      # left_knee
137                       # 26,      # right_knee
138                       # 27,      # left_ankle
139                       # 28,      # right_ankle
140                       # 29,      # left_heel
141                       # 30,      # right_heel
142                       # 31,      # left_foot_index
143                       # 32]      # right_foot_index
144                       ]
145
146     def __init__(self, root, dataTypes="All", testSubject="S2",
147                  token=None, categories=None,
148                  transform=None, pre_transform=None, pre_filter=None):
149
150         self.__processed = False
151
152         if dataTypes == "Dynamic":
153             self.StaticData = False
154             self.DynamicData = True
155         elif dataTypes == 'All':
156             self.StaticData = True
157             self.DynamicData = True
158         else:
159             self.StaticData = True
160             self.DynamicData = False
```

```
161
162         self.dataTypes = dataTypes
163         self.__testSubject = testSubject
164         self.alsoDownloadVideos = True
165
166         self.__token = token
167
168         if categories is not None:
169             categories = [gestures.lower() for gestures in categories]
170             for gesture in categories:
171                 if self.__categoriesStatic.__contains__(gesture) \
172                         and self.StaticData or \
173                         self.__categoriesDynamic.__contains__(gesture) \
174                         and self.DynamicData:
175                     self.__categories.append(gesture)
176         else:
177             if self.DynamicData and not self.StaticData:
178                 self.__categories = self.__categoriesDynamic
179             elif self.StaticData and not self.DynamicData:
180                 self.__categories = self.__categoriesStatic
181             else:
182                 self.__categories.extend(self.__categoriesStatic)
183                 self.__categories.extend(self.__categoriesDynamic)
184
185         self.number_targets = len(self.__categories)
186         self.number_nodes = len(self.__nodes_to_use)
187
188         # Video configurations
189         self.number_frames = 30
190         self.frames_gap = 3
191         self.frames_offset = 15  # 0,7 s at 22 fps
192
193         super().__init__(root, transform, pre_transform, pre_filter)
194         self.data, self.slices = torch.load(self.processed_paths[0])
195
196         if not self.__processed:
197             self.features = torch.load(
198                 os.path.join(self.processed_dir,
199                              f'{self.dataTypes[:3]}_feat.pt'))
200             self.targets = torch.load(
201                 os.path.join(self.processed_dir,
202                              f'{self.dataTypes[:3]}_trgs.pt'))
203             self.__test_features = torch.load(
204                 os.path.join(self.processed_dir,
205                              f'{self.dataTypes[:3]}_tsfeat.pt'))
206             self.__train_features = torch.load(
207                 os.path.join(self.processed_dir,
208                              f'{self.dataTypes[:3]}_trfeat.pt'))
209             self.__test_targets = torch.load(
210                 os.path.join(self.processed_dir,
211                              f'{self.dataTypes[:3]}_tstrgs.pt'))
212             self.__train_targets = torch.load(
213                 os.path.join(self.processed_dir,
214                              f'{self.dataTypes[:3]}_trtrgs.pt'))
```

```
215             self.__test_videos = torch.load(
216                 os.path.join(self.processed_dir,
217                              f'{self.dataTypes[:3]}_tsvids.pt'))
218             self.__train_videos = torch.load(
219                 os.path.join(self.processed_dir,
220                              f'{self.dataTypes[:3]}_trvids.pt'))
221             self.__videos = torch.load(
222                 os.path.join(self.processed_dir,
223                              f'{self.dataTypes[:3]}_vids.pt'))
224             self.CCO = torch.load(
225                 os.path.join(self.processed_dir,
226                              f'{self.dataTypes[:3]}_CCO.pt'))
227             self.__totalElements = torch.load(
228                 os.path.join(self.processed_dir,
229                              f'{self.dataTypes[:3]}_tels.pt'))
230 
231     @property
232     def raw_file_names(self):
233         # We look for two random files in order to decide if dataset
234         # needs to be downloaded.
235         if self.StaticData and not self.DynamicData:
236             raw_file_names_list = ['S1_attention_1_1m_upper.npy',
237                                    'S6_stop_2_4m_full.npy']
238             if self.alsoDownloadVideos:
239                 raw_file_names_list.append(
240                     os.path.join('videos',
241                                  'S1_attention_1_1m_upper.avi'))
242             return raw_file_names_list
243         elif self.DynamicData and not self.StaticData:
244             raw_file_names_list = ['S1_continue_3_6m_full.npy',
245                                    'S10_turnback_1_1m_upper.npy']
246             if self.alsoDownloadVideos:
247                 raw_file_names_list.append(
248                     os.path.join('videos',
249                                  'S1_continue_3_6m_full.avi'))
250             return raw_file_names_list
251         else:
252             raw_file_names_list = ['S1_attention_1_1m_upper.npy',
253                                    'S10_turnback_1_1m_upper.npy']
254             if self.alsoDownloadVideos:
255                 raw_file_names_list.append(
256                     os.path.join('videos',
257                                  'S1_continue_3_6m_full.avi'))
258                 raw_file_names_list.append(
259                     os.path.join('videos',
260                                  'S1_attention_1_1m_upper.avi'))
261             return raw_file_names_list
262 
263     @property
264     def processed_file_names(self):
265         # We generate a *.pt file with name composition of each gesture.
266         name = '_'.join([gesture[:2]
267                          for gesture in self.__categories])
268         return f'{self.dataTypes[:3]}' \
```

```
269                 f'_{str(self.alsoDownloadVideos)[:1]}' \
270                 f'_{name}.pt'
271
272     @property
273     def categories(self):
274         return self.__categories
275
276     def download(self):
277         git = Github(self.__token)
278         owner = git.get_user(self.__owner)
279         repository = owner.get_repo(self.__repo)
280         if os.path.exists(self.raw_dir):
281             shutil.rmtree(self.raw_dir)
282         os.makedirs(self.raw_dir)
283         if self.alsoDownloadVideos:
284             os.makedirs(os.path.join(self.raw_dir, "videos"))
285
286         self.__recursive_download(repository, self.__serverPath, self.raw_dir)
287
288     def __recursive_download(self, repository, server_path,
289                              local_path, content_prefix=""):
290         contents = repository.get_contents(server_path)
291
292         for content in contents:
293             if content.type == 'dir':
294                 # We use a RegEX to store subject folder recursively.
295                 prefix = content.name if re.search(
296                     "^S\d{0,}\$",
297                     content.name) else content_prefix
298                 self.__recursive_download(repository,
299                                           content.path, local_path, prefix)
300             elif content.type == 'file' and (
301                     ".npy" in content.name or (
302                     self.alsoDownloadVideos and ".avi" in content.name)):
303                 try:
304                     path = content.path
305                     if self.__categories.__contains__(
306                             content.name.split("_")[0]):
307                         if (self.StaticData
308                             and (("3Djoints" in path)
309                                  or ("videos" in path
310                                      and not ("dynamic_videos" in path)))) \
311                             or (self.DynamicData and ("dynamic" in path)):
312                             file_name = content_prefix + "_" + content.name
313                             file_content = repository.get_contents(path)
314                             file_data = base64.b64decode(file_content.content)
315                             if ".avi" in content.name:
316                                 file_out = open(os.path.join(
317                                     local_path, "videos", file_name), "wb")
318                             else:
319                                 file_out = open(os.path.join(
320                                     local_path, file_name), "wb")
321                             file_out.write(file_data)
322                             file_out.close()
```

```
323
324                 except (GithubException, IOError) as exc:
325                     print('Error downloading \%s: \%s', content.path, exc)
326
327     def process(self):
328         if os.path.exists(self.processed_dir):
329             shutil.rmtree(self.processed_dir)
330         os.makedirs(self.processed_dir)
331         if self.alsoDownloadVideos:
332             os.makedirs(os.path.join(self.processed_dir, 'videos'))
333
334         data_list = []
335         self.features = []
336         self.targets = []
337         self.__test_features = []
338         self.__train_features = []
339         self.__test_targets = []
340         self.__train_targets = []
341         self.__test_videos = []
342         self.__train_videos = []
343         self.__videos = []
344
345         # We create an extremely connected graph.
346         self.CCO = np.swapaxes([[i, j] for i in range(0, self.number_nodes)
347                                 for j in range(0, self.number_nodes)], 0, 1)
348
349         for gesture in self.__categories:
350             paths = glob.glob(os.path.join(self.raw_dir, f'*{gesture}*.npy'))
351             paths = sorted(paths, key=lambda e: (len(e), e))
352
353             for path in paths:
354                 is_test_subject = path.__contains__(self.__testSubject)
355                 gesture_seq = np.load(path, allow_pickle=True)
356                 number_of_sequences = ((gesture_seq.shape[
357                                             0] - self.frames_offset)
358                                        - self.number_frames)\
359                                       // self.frames_gap
360
361                 video_name = None
362                 input_video = None
363                 if self.alsoDownloadVideos:
364                     video_name = os.path.splitext(os.path.basename(path))[0]
365                     input_video = cv2.VideoCapture(
366                         f'{os.path.join(self.raw_dir, "videos", video_name)}'
367                         f'.avi')
368
369                 for seq in range(0, number_of_sequences):
370                     x = np.empty([self.number_nodes, 4, 0])
371                     init_frame = seq * self.frames_gap + self.frames_offset
372                     end_frame = init_frame + self.number_frames
373
374                     output_video = None
375                     output_video_path = None
376                     if self.alsoDownloadVideos:
```

```
377                         output_video_path = \
378                             f'{os.path.join(self.processed_dir, "videos",
                                ↪  video_name)}' \
379                             f'_{init_frame}' \
380                             f'_{end_frame}.avi'
381                         output_video = cv2.VideoWriter(
382                             output_video_path,
383                             cv2.VideoWriter_fourcc(*'XVID'),
384                             int(input_video.get(cv2.CAP_PROP_FPS)),
385                             (int(input_video.get(cv2.CAP_PROP_FRAME_WIDTH)),
386                              int(input_video.get(cv2.CAP_PROP_FRAME_HEIGHT))))
387
388                     for frame in range(init_frame, end_frame):
389                         pose = gesture_seq[frame,][0]
390                         frame_landmark = np.empty([0, 4])
391
392                         if self.alsoDownloadVideos:
393                             input_video.set(cv2.CAP_PROP_POS_FRAMES, frame)
394                             ret, video_frame = input_video.read()
395                             if ret:
396                                 mp.solutions.drawing_utils.draw_landmarks(
397                                     video_frame,
398                                     pose,
399                                     mp.solutions.pose.POSE_CONNECTIONS)
400                                 output_video.write(video_frame)
401
402                         # There's 33 landmarks in total.
403                         for landmark in range(0, 33):
404                             if self.__nodes_to_use.__contains__(landmark):
405                                 frame_landmark = np.append(
406                                     frame_landmark,
407                                     np.expand_dims(
408                                         [pose.landmark[landmark].x,
409                                          pose.landmark[landmark].y,
410                                          pose.landmark[landmark].z,
411                                          pose.landmark[landmark].visibility],
412                                         axis=0),
413                                     axis=0)
414
415                         # x = [n° nodes, 4, number_of_frames]
416                         x = np.append(x,
417                                       np.expand_dims(frame_landmark,
418                                                      axis=2), axis=2)
419
420                     if self.alsoDownloadVideos:
421                         output_video.release()
422
423                     x = np.swapaxes(x, 0, 1)
424                     # [4, number_nodes, number_of_frames]
425                     self.features.append(x)
426
427                     if is_test_subject:
428                         self.__test_features.append(x)
429                         self.__test_videos.append(output_video_path)
```

```
                    else:
                        self.__train_features.append(x)
                        self.__train_videos.append(output_video_path)

                    self.__videos.append(output_video_path)

                    x = np.swapaxes(x, 0, 2)
                    # [number_of_frames, number_nodes, 4]
                    x = torch.tensor(x, dtype=torch.float)

                    edge_index = torch.tensor(self.CCO)  # [2, 1089]
                    y = self.__categories.index(gesture)

                    target = np.zeros(len(self.__categories))
                    target[y] = 1
                    self.targets.append(target)

                    if is_test_subject:
                        self.__test_targets.append(target)
                    else:
                        self.__train_targets.append(target)

                    y = torch.tensor(y)  # [1]

                    data = Data(x=x, edge_index=edge_index,
                                edge_attr=None, y=y)
                    if self.pre_filter is not None\
                            and not self.pre_filter(data):
                        continue
                    if self.pre_transform is not None:
                        data = self.pre_transform(data)

                    data_list.append(data)

                if self.alsoDownloadVideos:
                    input_video.release()

        torch.save(self.collate(data_list), self.processed_paths[0])
        torch.save(self.features, os.path.join(
            self.processed_dir, f'{self.dataTypes[:3]}_feat.pt'))
        torch.save(self.__test_features, os.path.join(
            self.processed_dir, f'{self.dataTypes[:3]}_tsfeat.pt'))
        torch.save(self.__train_features, os.path.join(
            self.processed_dir, f'{self.dataTypes[:3]}_trfeat.pt'))
        torch.save(self.targets, os.path.join(
            self.processed_dir, f'{self.dataTypes[:3]}_trgs.pt'))
        torch.save(self.__test_targets, os.path.join(
            self.processed_dir, f'{self.dataTypes[:3]}_tstrgs.pt'))
        torch.save(self.__train_targets, os.path.join(
            self.processed_dir, f'{self.dataTypes[:3]}_trtrgs.pt'))
        torch.save(self.__test_videos, os.path.join(
            self.processed_dir, f'{self.dataTypes[:3]}_tsvids.pt'))
        torch.save(self.__train_videos, os.path.join(
            self.processed_dir, f'{self.dataTypes[:3]}_trvids.pt'))
```

```
484         torch.save(self.__videos, os.path.join(
485             self.processed_dir, f'{self.dataTypes[:3]}_vids.pt'))
486         torch.save(self.CCO, os.path.join(
487             self.processed_dir, f'{self.dataTypes[:3]}_CCO.pt'))
488 
489         self.__totalElements = len(data_list)
490         torch.save(self.__totalElements, os.path.join(
491             self.processed_dir, f'{self.dataTypes[:3]}_tels.pt'))
492 
493         self.__processed = True
494 
495     def get_dataset(self) -> Tuple[CustomDynamicGraphTemporalSignal,
496                                    CustomDynamicGraphTemporalSignal]:
497         """Returning the IRIGesture data iterator.
498 
499         Return types:
500             * **(train_dataset, test_dataset)**
501             *(tuple of DynamicGraphTemporalSignal)*
502              - The IRIGestureTemporalDataset.
503         """
504 
505         test_dataset = CustomDynamicGraphTemporalSignal(
506             self.__test_videos,
507             # List of CCO [2, self.number_nodes**2]
508             self.__get_edges(number_elements=len(
509                 self.__test_features)),
510             # List of ones (self.number_nodes**2, )
511             self.__get_edge_weights(number_elements=len(
512                 self.__test_features)),
513             # List each item (4, self.number_nodes, frames)
514             self.__test_features,
515             # List each item (frames, gestures)
516             self.__test_targets
517         )
518 
519         train_dataset = CustomDynamicGraphTemporalSignal(
520             self.__train_videos,
521             # List of CCO [2, self.number_nodes**2]
522             self.__get_edges(number_elements=len(
523                 self.__train_features)),
524             # List of ones (self.number_nodes**2, )
525             self.__get_edge_weights(number_elements=len(
526                 self.__train_features)),
527             # List each item (4, self.number_nodes, frames)
528             self.__train_features,
529             # List each item (gestures)
530             self.__train_targets
531         )
532 
533         return train_dataset, test_dataset
534 
535     def get_all_dataset(self) -> CustomDynamicGraphTemporalSignal:
536         """Returning the IRIGesture data iterator.
537 
```

```
538         Args types:
539             * **frames** *(int)* - The number of consecutive frames T,
540              default 16.
541         Return types:
542             * **dataset** *(DynamicGraphTemporalSignal)*
543              - The IRIGestureTemporal dataset.
544         """
545         dataset = CustomDynamicGraphTemporalSignal(
546             self.__videos,
547             # List of CCO [2, self.number_nodes**2]
548             self.__get_edges(),
549             # List of ones (self.number_nodes**2, )
550             self.__get_edge_weights(),
551             # List each item (4, self.number_nodes, frames)
552             self.features,
553             # List each item (gestures)
554             self.targets
555         )
556         return dataset
557 
558     def __get_edges(self, number_elements=None):
559         number_of_elements = self.__totalElements \
560             if number_elements is None else number_elements
561         edges = [self.CCO] * number_of_elements
562         return edges
563 
564     def __get_edge_weights(self, number_elements=None):
565         number_of_elements = self.__totalElements \
566             if number_elements is None else number_elements
567         edge_weights = [np.ones((
568             self.number_nodes ** 2,))] * number_of_elements
569         return edge_weights
570 
571     @property
572     def categories(self) -> List[str]:
573         return self.__categories
```

B4 *Tools.py*

```python
import os
import shutil
from typing import List, Tuple

import numpy as np
import pandas as pd
import seaborn as sn
import torch
import torchvision.io
from numpy import ndarray
from sklearn.metrics import confusion_matrix
from torch.backends import cudnn
from torch.utils.data import DataLoader, TensorDataset

from dataset.IRIDatasetTemporal import CustomDynamicGraphTemporalSignal


def seed_everything(seed: int = 1997):
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = True


def read_video(video_path: str) -> torch.Tensor:
    """
    Read a video with 4D tensor dimensions [time(frame),
     new_width, new_height, channel]
    and converts it to a 5D tensor [batch-size, time(frame),
     channel(color), height, width].
    """
    original_video = torchvision.io.read_video(video_path)
    video = np.transpose(original_video[0].numpy()
                         [..., np.newaxis], (4, 0, 3, 1, 2))
    return torch.from_numpy(video)


def create_confusion_matrix(y_pred, y_true, classes, title):
    # Build confusion matrix
    cf_matrix = confusion_matrix(y_true.cpu(), y_pred.cpu(),
                                 labels=[*range(len(classes))],
                                 normalize='true')
    df_cm = pd.DataFrame(cf_matrix / np.sum(cf_matrix) * 10,
                         index=[i for i in classes],
                         columns=[i for i in classes])
    if len(classes) > 8:
        sn.set(rc={'figure.figsize': (12, 8)})
    s = sn.heatmap(df_cm, annot=True)
    s.set(title=title)
    return s.get_figure()

```

```
53
54 def sort_list_by_indices(original_list: List,
55                           indices: List[int]) -> List[ndarray]:
56     return [i for _, i in sorted(zip(indices,
57                                      original_list))]
58
59
60 def create_data_loaders(dataset: CustomDynamicGraphTemporalSignal,
61                         batch_size: int, shuffle: bool,
62                         DEVICE: torch.device) -> DataLoader:
63     # Prepare features
64     features_array = np.array(dataset.features)  # (L, F, N, T)
65     features_transposed = np.transpose(
66         features_array, (0, 1, 3, 2))  # (L, F, T, N)
67     features_tensor = torch.from_numpy(
68         features_transposed).type(
69         torch.FloatTensor).to(DEVICE)  # (L, F, T, N)
70
71     # Prepare targets
72     target_array = np.array(dataset.targets)  # (L, G)
73     targets_values = np.argmax(target_array, axis=1)  # (L, )
74     target_tensor = torch.from_numpy(
75         targets_values).type(
76         torch.FloatTensor).to(DEVICE)  # (L, )
77
78     # Prepare videos
79     videos_list = np.linspace(0, len(dataset.videos_paths),
80                               len(dataset.videos_paths),
81                               False)  # (L, )
82     videos_tensor = torch.from_numpy(videos_list)\
83         .type(torch.IntTensor).to(DEVICE)  # (L, )
84
85     new_dataset = TensorDataset(features_tensor,
86                                 target_tensor,
87                                 videos_tensor)  # (L)
88     loader = DataLoader(new_dataset, batch_size=batch_size,
89                         shuffle=shuffle, drop_last=True)  # (L/B)
90
91     if len(loader) < 1:
92         raise AttributeError('Batch size is too big')
93     else:
94         return loader
95
96
97 def temporal_dataset_split(
98         data_iterator: CustomDynamicGraphTemporalSignal,
99         train_ratio: float = 0.8,
100 ) -> Tuple[CustomDynamicGraphTemporalSignal,
101            CustomDynamicGraphTemporalSignal]:
102     r"""Function to split a data iterator according to a fixed ratio.
103
104     Arg types:
105         * **data_iterator** *(Signal Iterator)* - Node features.
106         * **train_ratio** *(float)* - Graph edge indices.
```

```
107
108     Return types: * **(train_iterator, test_iterator)**
109      *(tuple of CustomDynamicGraphTemporalSignal Iterators)* -
110     Train and test data iterators.
111     """
112     train_snapshots = int(train_ratio * data_iterator.snapshot_count)
113
114     if type(data_iterator) == CustomDynamicGraphTemporalSignal:
115         train_iterator = CustomDynamicGraphTemporalSignal(
116
117             data_iterator.videos_paths[0:train_snapshots],
118             data_iterator.edge_indices[0:train_snapshots],
119             data_iterator.edge_weights[0:train_snapshots],
120             data_iterator.features[0:train_snapshots],
121             data_iterator.targets[0:train_snapshots],
122         )
123
124         test_iterator = CustomDynamicGraphTemporalSignal(
125             data_iterator.videos_paths[train_snapshots:],
126             data_iterator.edge_indices[train_snapshots:],
127             data_iterator.edge_weights[train_snapshots:],
128             data_iterator.features[train_snapshots:],
129             data_iterator.targets[train_snapshots:],
130         )
131     else:
132         raise TypeError('Must be CustomDynamicGraphTemporalSignal')
133
134     return train_iterator, test_iterator
135
136
137 def clear_path(path: str):
138     if os.path.exists(path):
139         shutil.rmtree(path)
140     os.makedirs(path)
```