UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Hand gestures recognition using 3D-CNN

A Degree Thesis

Submitted to the Faculty of the

Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona

Universitat Politècnica de Catalunya

by

Josep Famadas Alsamora

In partial fulfilment

of the requirements for the degree in

TELECOMMUNICATION SYSTEMS ENGINEERING

Advisor: Javier Ruiz Hidalgo

Barcelona, June 2017

Abstract

Since the emerge of informatic systems, one of the aspects that have helped to the rise of its popularity has been the simplification of the User-Computer communication, commonly known as user interface. Nowadays, the vanguard in this field are the techniques called touchless which, as its name indicates, consist of a kind of communication which do not imply touching any sort of hardware, by means of audio or video.

This project involves the recognition of dynamic hand gestures performed with hands using RGB-D (Color and Depth) sequences recorded with a Kinect sensor. In order to do so I have used a technique which combines computer vision and deep learning known as 3D Convolutional Neural Network.

My solution is inspired in the one proposed by *Molchanov et al* in their work [1] where some spatial and temporal data augmentation techniques have been used.

In my case I have worked with two different datasets.

The first one is a prepared dataset. With it, an accuracy of nearly 65% has been obtained.

The second one (which will be named as Telepresence Dataset) has been self-made. With it, I did not get positive results.

Resum

Des de l'aparició dels sistemes informàtics, un dels aspectes que han ajudat a l'augment de la seva popularitat ha estat la simplificació de la comunicació Usuari-Ordinador, altrament coneguda com interfície d'usuari. Actualment l'avantguarda d'aquest camp es troba en les tècniques conegudes com a *touchless* que, tal i com el seu nom indica, consisteixen en una comunicació que no impliqui tocar cap hardware, ja sigui mitjançant àudio o vídeo.

En aquest projecte treballo el reconeixement de gestos dinàmics fets amb les mans utilitzant seqüències RGB-D gravades amb un sensor Kinect. Per dur a terme això he utilitzat una tècnica que combina *computer vision* i *deep learning* coneguda com a Xarxa Neuronal Convolucional 3D.

La meva solució està inspirada en la que proposen *Molchanov et al* en el seu treball [1] on s'utilitzen tècniques de *data augmentation* tant temporal com espacialment.

En el meu cas he treballat amb dos *datasets* diferents.

El primer estava preparat. Amb ell, s'ha aconseguit un encert de quasi 65%.

El segon (Al qual em referiré com a *Telepresence Dataset*) ha estat creat per mi. Amb ell, no he obtingut resultats positius.

Resumen

Desde la aparición de los sistemas informáticos, uno de los aspectos que han ayudado más al aumento de su popularidad ha sido la simplificación de la comunicación Usuario-Ordenador, también conocida como interfaz de usuario. Actualmente la vanguardia de este campo se encuentra en las técnicas conocidas como *touchless* que, tal y como su nombre indica, consisten en una comunicación que no implique tocar ningún hardware, ya sea mediante audio o video.

En este proyecto trabajo el reconocimiento de gestos dinámicos hechos con las manos usando secuencias RGB-D grabadas con un sensor Kinect. Para llevar eso a cabo he usado una técnica que combina computer vision y deep learning conocida como Red Neuronal Convolucional 3D.

Mi solución está inspirada en la propuesta por *Molchanov et al* en su trabajo [1] donde son usadas técnicas de *data augmentation* tanto temporal como espacial.

En mi caso he trabajado con dos *datasets* distintos.

El primero estaba preparado. Con él, he conseguido un acierto de casi 65%

El segundo (Al cual me referiré como *Telepresence Dataset*) ha sido creado por mí. Con él, no he obtenido resultados positivos.

To my family and Núria who, since the day I started the degree, always thought I would make it.

Acknowledgements

I would like to acknowledge Javier Ruiz not only for being my project advisor and helping me when I was stacked but also for introducing me to the fascinating world of machine learning and deep learning.

I would also like to acknowledge the members of telepresence group which, in one way or another, have helped me to reach the conclusion of the project.

I want to thank especially all those people who spend one part of their time to become subjects of the Telepresence Dataset.

Despite the fact that they have not collaborated directly with this project, I would like to acknowledge all the ETSETB professors which have helped me through the degree to arrive here and reach what I have reached.

Finally, I would like to do a mention to the YouTube channel *codigofacilito* which, with his course [2], has helped me to learn Python from scratch to be able to work with TensorFlow.

Revision history and approval record

Revision	Date	Purpose
0	06/06/2017	Document creation
1	21/06/2017	Document revision
2	30/06/2017	Document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Josep Famadas Alsamora	jfamadas95@gmail.com
Javier Ruiz Hidalgo	j.ruiz@upc.edu

Written by:		Reviewed and approved by:	
Date	06/06/2017	Date	30/06/2017
Name	Josep Famadas Alsamora	Name	Javier Ruiz Hidalgo
Position	Project Author	Position	Project Supervisor

Table of contents

List of Figures

List of Tables:

1. Introduction

Since the emerge of informatic systems, one of the aspects that have helped to the rise of its popularity has been the simplification of the User-Computer communication, commonly known as user interface.

This simplification is reached by means of making the communication as natural as possible for a human being. It started with the command line interface, which was closed and a very strict way with limited possibilities. After it, the graphical user interface appeared, with the desktop and the icons system which were a more intuitive way of communicating. After that, smart screens were a revolution because they eliminated the mouse and the keyboard, making it even more easy.

As seen in the *Figure 1*, this evolution is made by eliminating intermediaries until we reach a hypothetic future point in which we will be our own computer system.

Nowadays, the vanguard in this field are the techniques called touchless which, as its name indicates, consist of a kind of communication in which you do not need to touch the hardware, you can communicate with it by speech or gestures.

Communicating to a system by gestures is a really natural and intuitive because it gets more similar to what humans do to communicate to each other.

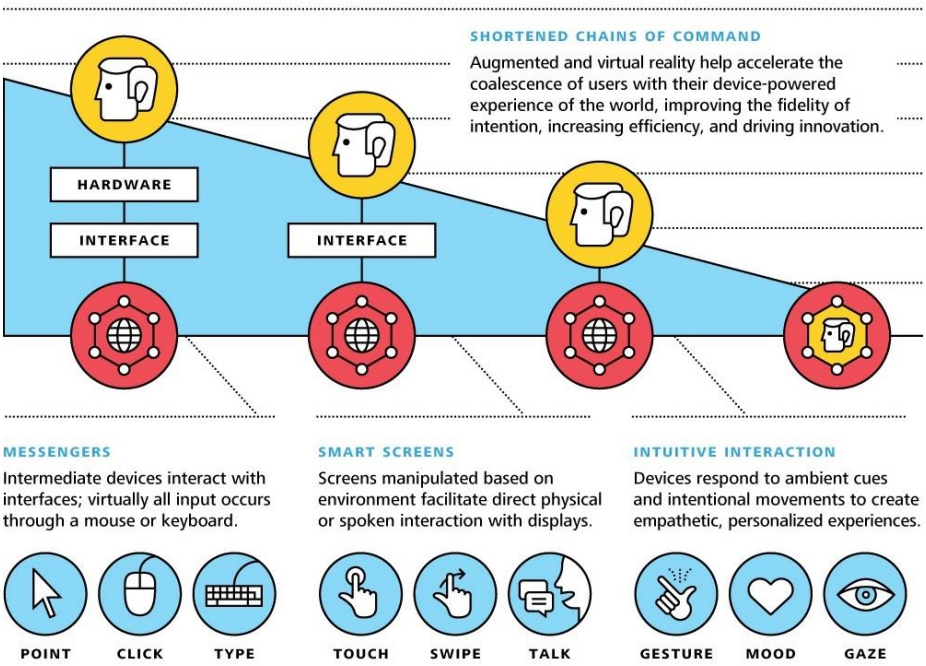

Figure 1: Evolution of User-Computer interaction[3]

But this “eliminated” element do not magically disappear, they can be removed because their function can be replaced by a software, and it has not been since early 2000s that the computational power has been enough to make viable a way to train computers so they can not only recognize your gestures or voice, but also actuate in consequence.

The state of the art of these ways is the machine learning, which will be described in this project so it is the technique that has been used on it.

1.1. Project background

This project is part of the “telepresence” project carried out by a group of master and degree students in the Signal Theory and Communications department with the supervision of the professors Javier Ruiz Hidalgo and Josep R. Casas and the technician Albert Gil Moreno.

The whole project consists on making a 3D representation of a room and its content (by means of 3 Kinect sensors placed in different sites of the room) and bring it to a HTC VIVE virtual reality headset.

The idea of hand gesture recognition was provided by Javier Ruiz Hidalgo who proposed me to get some ideas from a previous work done using the technique of random forests [4] and to use deep learning techniques instead.

1.2. Statement of purpose

The project has been carried out at the UPC, at the Signal Theory and Communications department.

This project mainly consists on creating a software able to detect hand gestures performed in front of a Kinect device and interpret them.

To sum up, the main goals of my project are:

1. Design a system able to detect hand gestures in RGB-D images by means of the state of the art deep learning techniques.
2. Train this system using a previously decided dataset in order to have an acceptable ratio of detection and low ratio of false alarms
3. Create my own dataset in order to integrate the system in the Telepresence project.
4. Retrain the system with the new dataset and finally integrate it to the big project.

1.3. Requirements and specifications

Project requirement:

- Given a RGB-D sequence the system must be able to detect the gesture performed in this sequence.

Project specification:

- Have a detection probability higher enough and a false alarm probability lower enough to consider it a robust and reliable system.

1.4. Workplan

<table>
<tr><td>Project: hand gesture recognition algorithm design</td><td colspan="2">WP ref: 1</td></tr>
<tr><td>Major constituent: Design of the algorithm</td><td colspan="2">Sheet 1 of 6</td></tr>
<tr><td rowspan="2">Short description:
Design a CNN (inspired in the one created by Molchanov et al.) able to detect and recognize hand gestures in a video recorded with a Kinect sensor.</td><td colspan="2">Planned start date: 15/02/2017
Planned end date: 15/03/2017</td></tr>
<tr><td colspan="2">Start event:
End event:</td></tr>
<tr><td>Internal task T1: Research on the state of the art in hand gesture recognition
Internal task T2: Design schematically the algorithm</td><td>Deliverables:
(in milestones table)</td><td>Dates:
(in milestones table)</td></tr>
</table>

<table>
<tr><td>Project: Hand gesture recognition algorithm implementation</td><td colspan="2">WP ref: 2</td></tr>
<tr><td>Major constituent: Implemented algorithm</td><td colspan="2">Sheet 2 of 6</td></tr>
<tr><td rowspan="2">Short description:
Choose a framework (MatLab, Tensorflow, Caffe, Keras) to work with and implement the designed algorithm in the respective code language.</td><td colspan="2">Planned start date: 08/03/2017
Planned end date: 02/04/2017</td></tr>
<tr><td colspan="2">Start event:
End event:</td></tr>
<tr><td>Internal task T1: Research on the best framework to work with
Internal task T2: Implement the designed algorithm</td><td>Deliverables:
(in milestones table)</td><td>Dates:
(in milestones table)</td></tr>
</table>

<table>
<tr><td>Project: Training and testing the CNN</td><td colspan="2">WP ref: 3</td></tr>
<tr><td>Major constituent: Software training and testing</td><td colspan="2">Sheet 3 of 6</td></tr>
<tr><td rowspan="2">Short description:
Find a dataset to train our CNN applying (if necessary) data extension techniques.
Once the CNN is trained, test it and verify if it fulfils the specifications.</td><td colspan="2">Planned start date: 08/03/2017
Planned end date: 27/04/2017</td></tr>
<tr><td colspan="2">Start event:
End event:</td></tr>
<tr><td>Internal task T1: Research in order to find the best dataset in terms of type of gestures, amount and accessibility.
Internal task T2: Train the CNN using the decided dataset and find the weights of it.
Internal task T3: Test the trained CNN and verify the specifications.
Internal task T4: Since it does not work appropriately, fix the CNN</td><td>Deliverables:
(in milestones table)</td><td>Dates:
(in milestones table)</td></tr>
</table>

Project: 3 Kinect sensor adaptation	WP ref: 4	
Major constituent: Software modification	Sheet 4 of 6	
Short description: Modify the software in order to be able to take the image from 3 sensors instead of 1 placed in different sites of a room.	Planned start date: 06/04/2017 Planned end date: 03/05/2017	
	Start event: End event:	
Internal task T1: Design an algorithm to take the image from the 3 sensors and decide. Internal task T2: Implement the designed algorithm. Internal task T3: Test the algorithm. Internal task T4: Dataset Creation	Deliverables: (in milestones table)	Dates: (in milestones table)

Project: Real time data	WP ref: 5	
Major constituent: Final software	Sheet 5 of 6	
Short description: Integrate totally my project with the general telepresence project and test its performance with real time data got from another mate's part	Planned start date: 04/05/2017 Planned end date: 31/05/2017	
	Start event: End event:	
Internal task T1: Final integration Internal task T2: Final test	Deliverables: (in milestones table)	Dates: (in milestones table)

Project: Project Documentation	WP ref: 6	
Major constituent: Final software	Sheet 6 of 6	
Short description: Documentation of my final degree project.	Planned start date: 15/02/2017 Planned end date: 30/06/2017	
	Start event: End event:	
Internal task T1: Proposal and Workplan Internal task T2: Critical design review Internal task T3: Final report	Deliverables: (in milestones table)	Dates: (in milestones table)

1.5. Gantt diagram

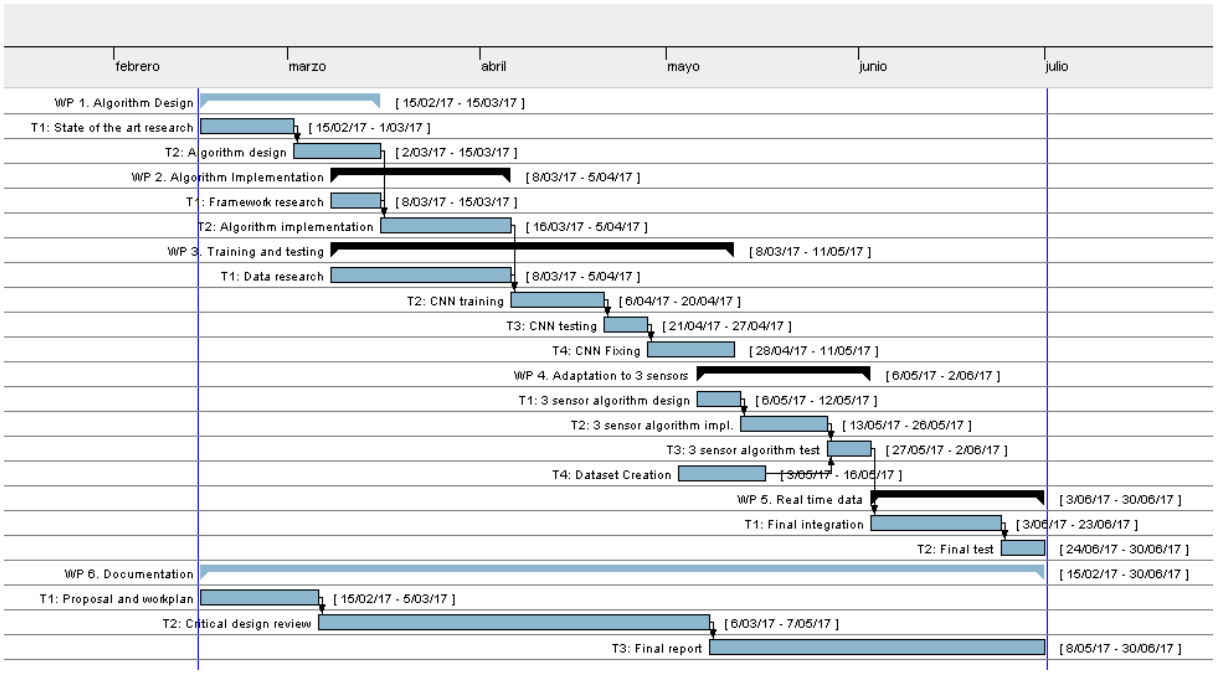

Figure 2: Original Gantt diagram

1.6. Incidences

Overall the first 3 work packages were carried out without any noticeable incidence. Due to a lack of time, the part of the final integration with the Telepresence big project (WP 5) was eliminated and delayed for a hypothetical future work. WP 4 was made with just one of the three sensors.

All in all, in this project there has been a major incidence, which is the fact that in the part of the second dataset (WP4) I did not get the expected results. As I show later on the results part, while with the VIVA dataset the cost function kept decreasing until reaching 0 (and, as a consequence, the accuracy increasing) with the Telepresence dataset it reached a point where it got stuck and there was no way to decrease it. I will explain in the results part all the solutions I have tried.

Apart of this, there was not any other remarkable incidence.

2. State of the art

2.1. Deep Learning

With a first shallow look on the machine learning we realize that the state of the art are the Artificial Neural Networks (ANN).

ANN are computing systems that simulates our neural system. They are based on a collection of connected units called artificial neurons which are organized in layers as seen in *Figure 3*, where the first is the Input Layer (Yellow), the last is the Output Layer (Reddish) and all the other are known as Hidden Layers (Blue). As it can be seen, in a neural network each neuron is fully-connected (which means that is connected to all the neurons in the previous layer and to all the neurons in the next layer).

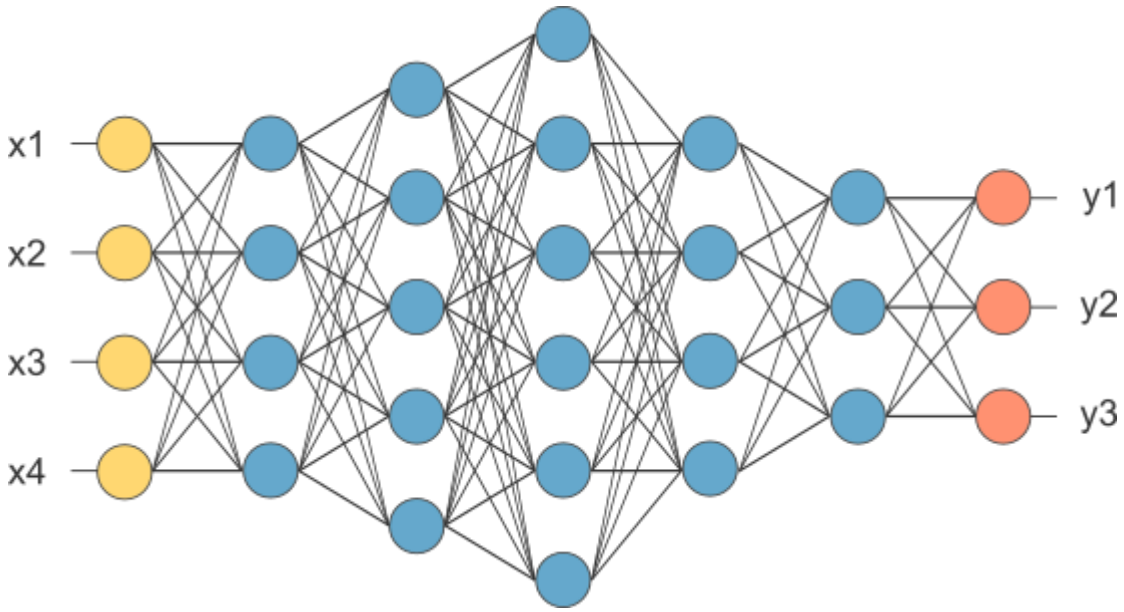

Figure 3: Image of an Artificial Neural Network [5]

The hidden layer neurons follow all the same functionality steps:

1. Receive a real number as input from another neuron.
2. Multiply it by a weight (**W**) and add a bias (**b**) to it. One neuron has one pair of **W** and **b** associated to each of the neurons that give an input to it.
3. Apply a nonlinear operation to the result to break the linearity. Here we can typically see functions such as Rectified Linear Unit (ReLU) or Sigmoid function.
4. Send the current value to all the neurons connected to its output.

The output layer is interpreted as a one-hot sequence; the neuron with the higher output value is interpreted as a 1 and all the others as 0. So, the decision of the classification is the one associated with that neuron.

One practical example is when we want to know if in an image there is a car, a motorcycle or a truck. The neurons of the input layer would be each one of the pixels of the image and the output layer would have 3 neurons. One associated with each decision (car, motorcycle or truck).

The point of the neural network is the training, where you show it some training examples while telling it the correct decision of each one of the examples. Through this process, by means of algebraic algorithms, the network tunes all its **W**s and **b**s in order to give the correct result when shown the same examples again. If the amount of training examples is enough big you will reach the point when you can show the network an example it has never seen but it will give the correct answer.

The advantage of my project is that I know *a priori* that i will be working with images, so we can narrow the range of usable techniques to reach the top ones which are the Convolutional Neural Networks (CNN), a variation of ANN specifically designed for image analysis.

2.2. Convolutional Neural Networks

Convolutional Neural Networks are very similar to Artificial Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and follows it with a non-linearity.

To understand the difference, imagine a classificator of 640x480x3 (Height x Width x #Channels (RGB in this case)) images as input and a first hidden layer being a fully-connected layer of 10 neurons. This will mean 9,210,000 weights!

This number is totally unfeasible so that's why CNNs, making the assumption that the inputs are images, add two new types of layers: Convolutional layer and Pooling layer.

- Each Convolutional Layer consists on a group of trainable filters (known as kernels). Every filter is small spatially (along width and height), but extends through the full depth (Number of channels) of the input volume. The output of a Convolutional Layer is obtained by convoluting each of the kernels with the input volume. It is a volume spatially smaller than the input one and with the same depth as the number of filters. Following the previous example a Convolutional Layer could be formed by 8 kernels 5x5x3.
- The Pooling layers reduce the spatial size of its input volume in order to reduce the number of parameters and computation in the network.

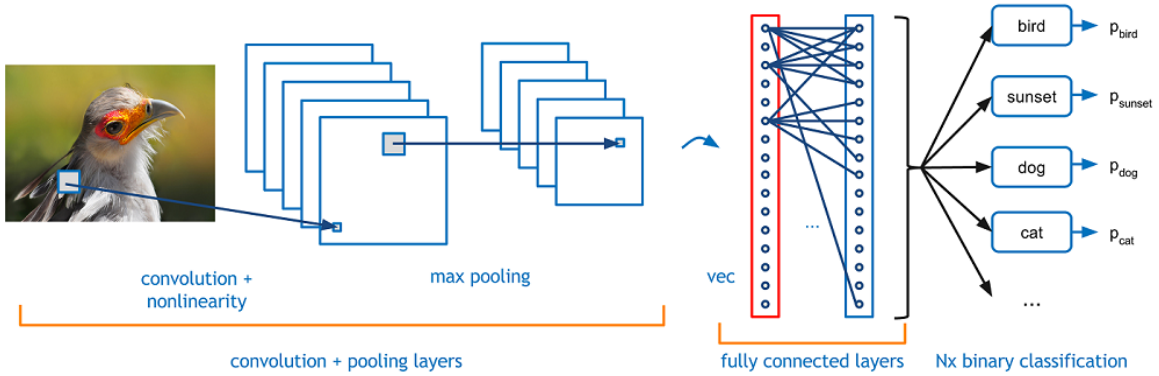

Figure 4: (from left to right) An image, a convolutional layer, a max pooling layer, two fully connected layers and the decision layer [6]

For more information, here [7] is a full and deeper explanation about CNN.

But, returning to my project, we can specify even more. Given the fact that the project consists on analyzing **dynamic** gestures the objective of our network will be not only finding spatial patterns (High x Width x Depth) but also temporal patterns.

In order to do so, there are two different techniques that are part of the state of the art: 3D Convolutional Neural Networks (3D-CNN) and Recurrent Neural Networks (RNN).

2.3. 3D CNN - vs - Recurrent Neural Networks

On the one hand, Recurrent Neural Networks (RNN) can be defined as Neural Networks with memory. The idea behind RNNs is to make use of sequential information. In a traditional neural network, we assume that all inputs and outputs are independent of each other, but in my project that assumption is totally erroneous because each frame of a sequence is strongly correlated with the previous ones.

The working method of a RNN can be seen as if you add a hidden short time memory between layers of Neural Network. In this way, with the next item of the sequence, you do not only feed the network with it but also with the result of the previous item so now the training takes into account the relation between both.

RNNs shine when working with speech or text recognition because they are not limited to a fixed number of inputs in a sequence.

On the other hand, knowing that we can fix the number of frames per sequence, we have the 3D-CNN which are, as its names indicates, a 3-dimensional version of ordinary CNN

where the Kernels and the input and output volumes have on extra dimension (time dimension) and the convolution is performed through width, height and time.

This fact makes 3D-CNN really well-suited for spatiotemporal feature learning. Compared to ordinary CNN, 3D-CNN have the ability to model temporal information better owing to 3D convolution and 3D pooling operations. Despite the fact that ordinary Convolutional Layers could also work with video sequences (if you treat the frames as channels) they would give an image as output so you would lose the temporal information. In 3D-CNN the channels and the temporal axis are treated separately meaning that with a volume input you give a volume output, avoiding to lose the temporal information.

For my project, I have chosen 3D-CNN because, as Du Tran et al. demonstrated [8], compared with Recurrent Neural Networks based methods, 3D-CNNs outperform Long-term Recurrent Convolutional Networks (LRCN) and LSTM composite model by 14.1% and 9.4% respectively. They are more specifically designed for video recognition so I thought they were more suitable for my commitment.

2.4. RGB-D Images in gesture recognition

Taking an RGB-D image of something is a way of having more information about it. Explicitly, this kind of images have an extra channel (apart of the red, green, and blue that describe the color) which corresponds to the distance of each pixel to the sensor.

This already sounds great, but when it comes to object recognition (or gestures in my case) it's marvelous. The depth channel allows you to have extremely useful information for image processing. For example, it makes it really easy for a machine learning system to detect the edges of objects and ignore the background and focus just on them.

In my case, working with gestures performed by a person, RGB-D video sequences make it easier for my system to detect the person performing the gesture and focus on his hands.

Currently, the state of the art in RGB-D sequences are the Microsoft Kinect sensors which use the time-of-flight technique. It is performed by sending a mesh of infrared points and capture the returning mesh calculating the time that each point has delayed to reach the sensor, transforming this information to distance.

3. Methodology / project development:

Before starting to design the deep learning system some previous work needed to be done. First of all, as mentioned before, to train a deep learning network you need a dataset, so selecting the dataset I would work with was a must. After that, the dataset could not be feet raw to the network, it needed to be prepared with a pre-processing.

3.1. Pre-deep learning

3.1.1. Dataset

In this project, I have worked with two different datasets:

- VIVA Challenge [9]: Dataset recorded using a Kinect device inside a vehicle. There are 19 different hand gestures, performed by 8 subjects from 3 to 6 times each. A total amount of 885 sequences with a spatial resolution of 250 x 115 and different number of frames.

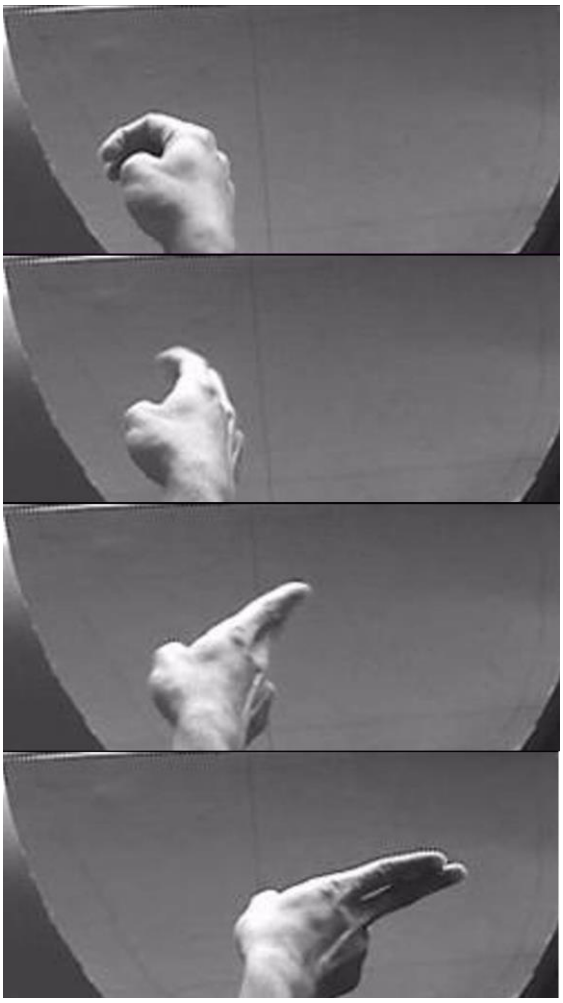

Figure 5: Frames from the first gesture of the VIVA Challenge dataset

- Telepresence: Dataset recorded by Albert Gil and me in the ETSETB smart room. There are 6 gestures, performed by 9 subjects 5 times each. A total amount of 270 sequences with a spatial resolution of 512 x 424 and different number of frames.

Figure 6: Frames from the first gesture of the Telepresence dataset

3.1.2. Pre-processing

First of all, due to the fact that every sequence has a different number of frames, I normalized the sequences duration to 32 frames. I did this by repeating or erasing equispaced frames.

Secondly, I transformed only the RGB frames into Grayscale and applied a Sobel filter in order enhance the system robustness against different illumination.

Finally, I normalized both channels of each sequence (Grayscale and Depth) to have zero mean and unit variance so the system could converge faster and then resized them to be 125 x 57 so they can fit in the 3D-CNN.

Once the whole dataset was processed I saved all the gestures in separated NumPy [10] files with shape [number of sequences, 32, 57, 125, 2].

This preprocessing was applied to both datasets.

3.2. Deep learning system

My whole system consists mainly in two 3D-CNN called High Resolution Network (HRN) and Low-Resolution Network (LRN). Both networks produced their own class-probability at the SoftMax layer and they are multiplied with the ones from the other subnetwork providing a final decision.

The HRN consists of 4 Convolutional Layers each one followed by a Max-pooling Layer. The parameters can all be seen in *Figure 7* The output volume of the fourth Max-pooling Layer is connected to a Fully Connected layer with 512 neurons followed by another one with 256. Finally, there is the SoftMax layer which has 19 neurons corresponding to the 19 different gestures. When working with the Telepresence dataset the SoftMax layer has only 6 neurons.

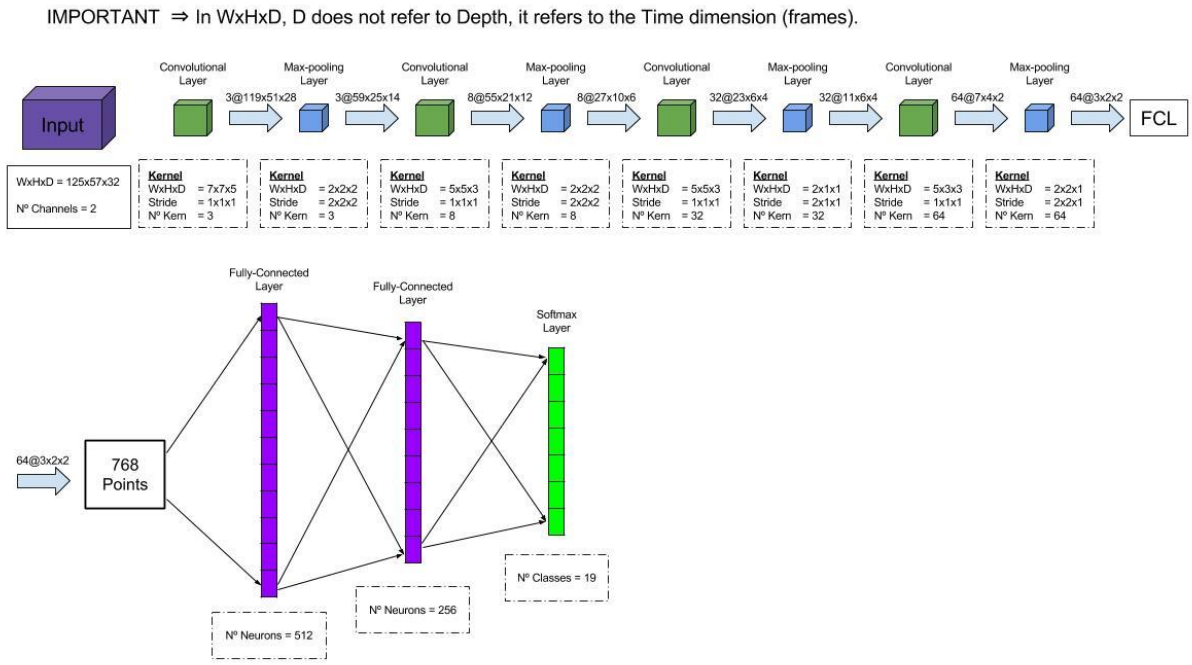

Figure 7: High Resolution Network graphical design

The LRN consists on a first down sampling layer which reduces the height and width of the input volume by a factor of 2. Then, there are 3 Convolutional Layers each one followed by a Max-pooling Layer. The parameters can all be seen in *Figure 8*. The output volume of the third Max-pooling Layer is connected to a Fully Connected layer with 512 neurons followed by another one with 256. Finally, there is the SoftMax layer which has 19 neurons corresponding to the 19 different gestures. When working with the Telepresence dataset the SoftMax layer has only 6 neurons.

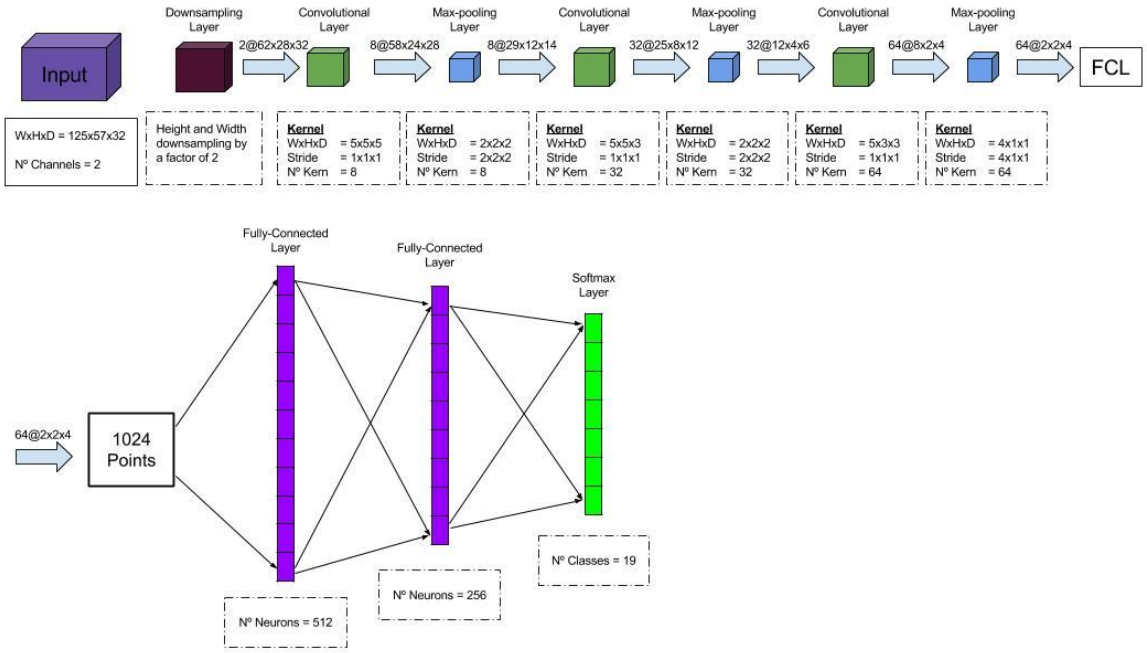

Figure 8: Low Resolution Network graphical design

Every layer of both sub-networks except for the SoftMax ones if followed by a ReLU function to break the linearity.

3.3. Training

Before starting to train the classifier, three data augmentation techniques were used in order to increase the dataset size and help it to avoid overfitting. The first one was a horizontal mirror effect, the second one was reversing the sequence in time, which means a mirror effect in the 32 frames, and the last one was a combination of the first two. Thanks to this I could multiply by 4 the amount of sequences in the training dataset.

The first step of the training is the parameters initialization. I initialized the weights of the 3D-CNN with following *Table 1*.

	Initialization Type	Value
Convolutional layer Weights	Random (uniform)	[-K, K] $K = \sqrt{\frac{6}{\#\ Input\ neurons + \#\ Output\ neurons}}$
Fully-connected layer Weights	Random (normal)	mean = 0 std = 0.01
Biases (except SoftMax layer)	Constant	1
Biases (SoftMax layer)	Constant	0

Table 1: Initialization of the network parameters

For the training process, I have decided to use as cost function the multiclass cross entropy function. The training is performed using the stochastic gradient descent method with a learning rate of 0.005 and with mini-batches of 20 and 40 for the HRN and LRN respectively. Both sub-networks are trained separately.

Due to the fact of not having a test set I decided to use the leave-one-out technique which consisted on doing the training with all the sequences except the ones from 1 of the subjects and use them as test set.

4. Results

I evaluated my hand gesture recognition system with the two different datasets, obtaining the following results:

4.1. VIVA Dataset

With the VIVA Dataset, I must say that the results were pretty satisfying. All the results were performed with an amount of 20,000 iterations. *Table 2* shows the final and overall results when using just the HRN, just the LRN and both, with and without data augmentation.

First of all, as expected, applying data augmentation helps with the generalization from the training set to the test set, so that is why the trainings using that technique have a better accuracy.

As it can be observed, The Low Resolution Network is far better than the High Resolution Network. This fact can shock a little bit at the beginning; why is it better the LRN if the HRN uses more precise data to train? The reason of this is the over fitting: using more precise data it learns better to recognize the training set but it is harder for it to generalize then to the training set.

Finally, as it was expected, the combination of both networks has the best performance.

	HRN	LRN	HRN + LRN
Without Data Augmentation	52.63 %	57.89%	61.40 %
With Data Augmentation	56.33 %	59.68 %	64.91 %

Table 2: VIVA Challenge training set accuracy

Figure 9 and *Figure 10* show the confusion matrix numerically and in a colormap respectively. The confusion matrix is a very common technique used for evaluation of errors in predictions. It consists on a matrix where columns represent the predicted case and rows represent the actual case. The ideal case would be if just the diagonal of the matrix had numbers, what would mean a 100% of accuracy.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
A[	19	12	0	0	1	0	0	2	4	0	0	0	0	0	3	0	1	0	0]
B[	13	18	0	0	1	0	1	2	3	1	0	0	2	0	0	0	0	1	0]
C[	0	0	22	6	0	0	0	0	0	2	2	0	0	3	1	2	0	2	2]
D[	0	0	5	18	0	0	0	0	0	6	2	3	0	2	0	0	4	2	0]
E[	2	2	2	1	29	3	1	0	0	0	0	0	0	0	1	1	0	0	0]
F[	0	1	0	0	3	22	8	0	0	1	1	0	0	0	1	0	3	0	2]
G[	1	0	3	0	0	3	25	1	0	0	0	4	2	0	0	0	2	1	0]
H[	1	3	0	0	0	0	0	28	9	0	0	0	1	0	0	0	0	0	0]
I[	4	4	0	0	0	0	0	14	18	0	1	0	0	0	1	0	0	0	0]
J[	0	0	2	3	1	0	1	0	0	25	8	0	0	2	0	0	0	0	0]
K[	0	0	0	2	0	0	0	0	0	13	27	0	0	0	0	0	0	0	0]
L[	1	0	1	3	0	0	1	0	0	1	2	26	3	4	0	0	0	0	0]
M[	0	2	0	0	0	1	0	1	1	1	0	2	17	6	5	0	2	0	4]
N[	0	0	2	3	1	0	0	0	0	0	0	7	1	18	6	2	1	0	1]
O[	2	0	3	2	0	0	0	0	0	0	0	0	2	5	23	3	2	0	0]
P[	0	1	2	0	0	0	2	0	0	0	0	1	0	2	2	22	9	1	0]
Q[	1	0	3	2	1	0	0	0	0	0	0	1	0	3	3	14	11	0	3]
R[	0	1	3	0	0	0	0	0	0	0	0	0	0	1	1	2	5	20	9]
S[	0	0	6	0	0	0	0	0	0	0	0	0	1	2	3	0	4	9	17]

Figure 9: Confusion matrix (Numerical)

In *Figure 9* the results can be easily appreciated. While it is true that the diagonal is the most brightening part of the matrix there can be seen that there appear some kind of 2x2 squares on it. That is because the gestures where organised by pairs, I mean, gesture A is “Swipe Left” and gesture B is “Swipe Right”, so that is why they are easily confused by the network.

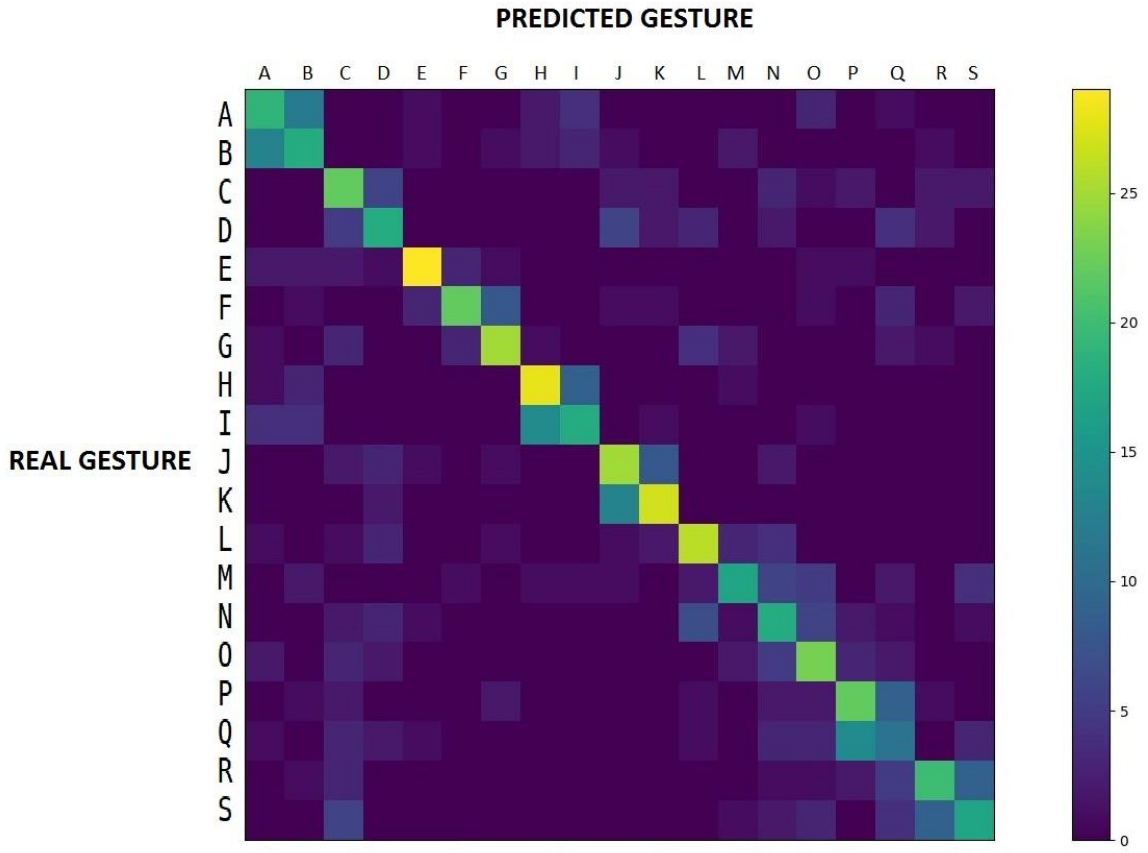

Figure 10: Confusion matrix (Colormap)

Figure 11 shows the cost function calculated with the training set and with the test set compared to the number of epochs. As we can see, when we reach approximately 7,500 epochs, the test cost function stops decreasing and saturates. This point is one of the most important in deep learning, it marks the start of the system over fitting, what means that even if the training accuracy keeps increasing the network is no more capable of generalize the results to no-training data. As it can be seen in *Figure 12* once the saturation point has been reached, the test set accuracy gets also saturated.

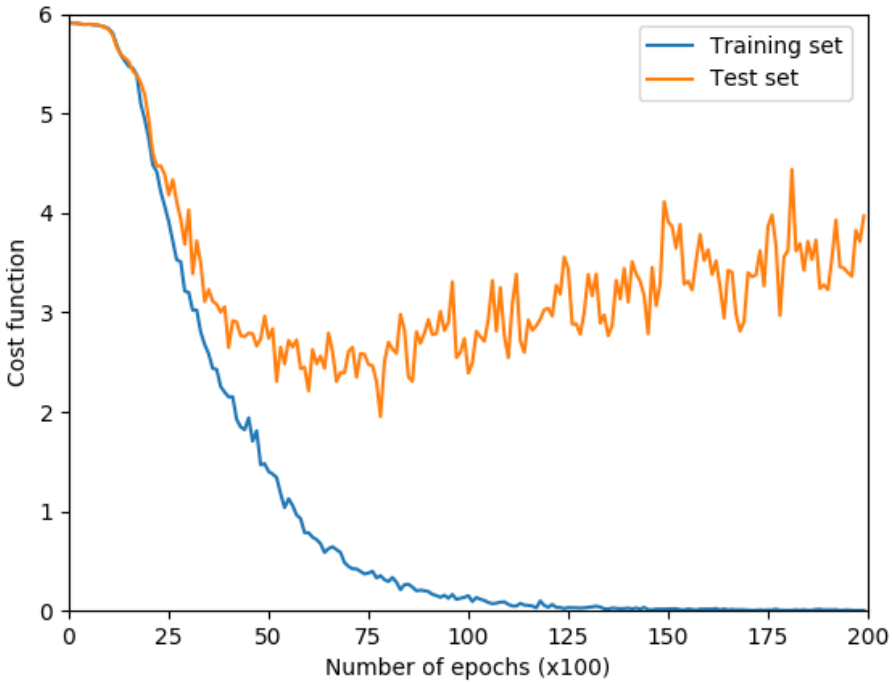

Figure 11: Cost function evaluated with the training set (blue) and test set (orange) in function of the number of epochs

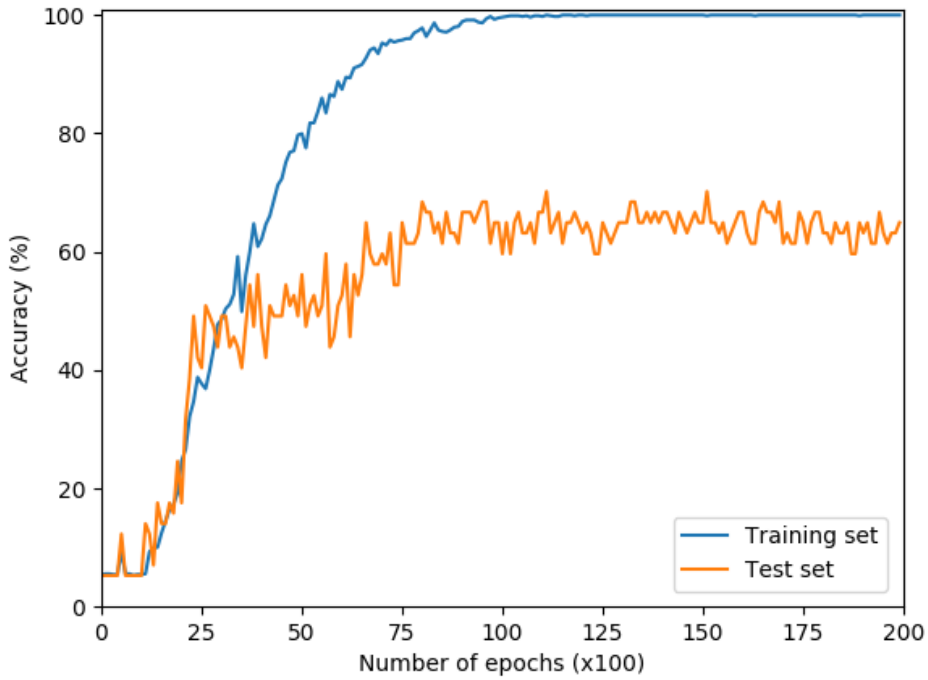

Figure 12: Percentage of correctly predicted gestures from the training set (blue) and from the test set (orange) in function of the number of epochs

4.2. Telepresence Dataset

As it was introduced in the incidences part, with the Telepresence Dataset the results were not the ones I expected them to be. Despite trying a lot of different methods, the cost function did saturate really early and the accuracy did not increase.

First of all, to discard the possibility of being a problem of not enough epochs, it was made a training with 64,000. However, as it can be seen in *Figure 13,* the cost function had the same value from the iteration 5,000 to the end.

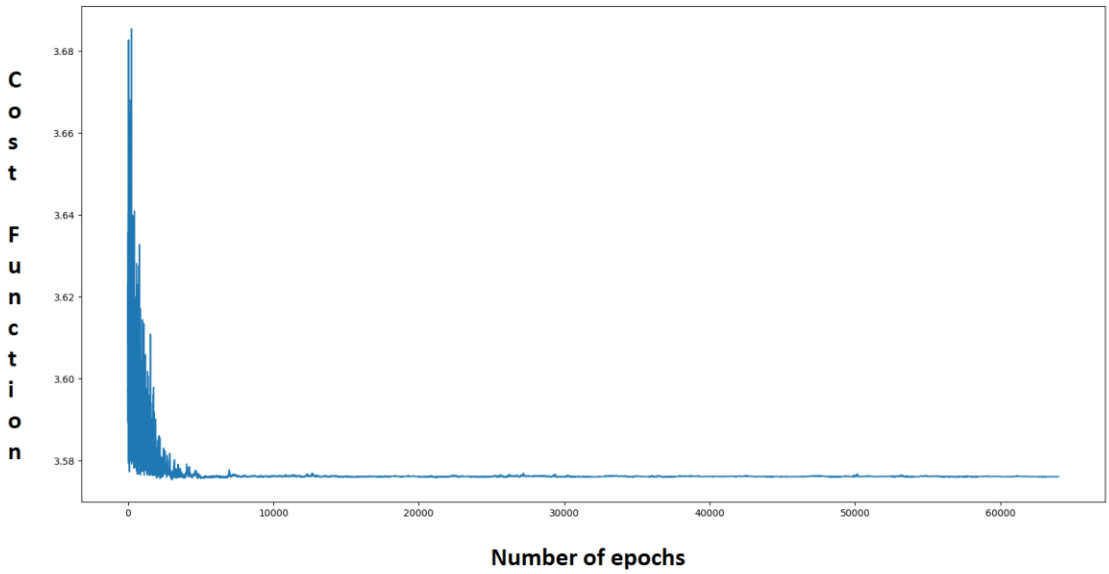

Figure 13: Cost function evaluated with the Telepresence dataset with the full network

As a second possibility, I thought of changing the learning rate, trying to train the network with a bigger one or with a smaller one. As can be seen in *Figure 14*, *Figure 15* and *Figure 16,* changing the learning rate just helps it to reach the saturation point faster (in case it is increased) or slower (in case it is reduced).

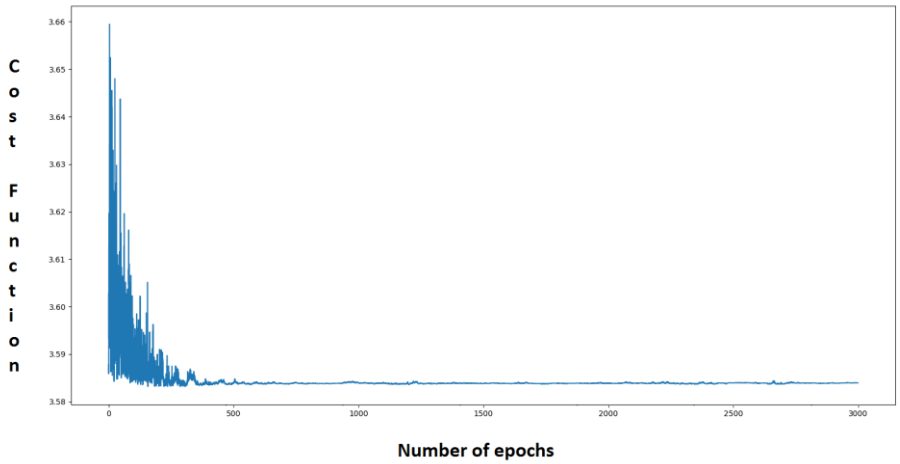

Figure 14: Cost function evaluated with the Telepresence dataset with the full network and a learning rate of 0.01

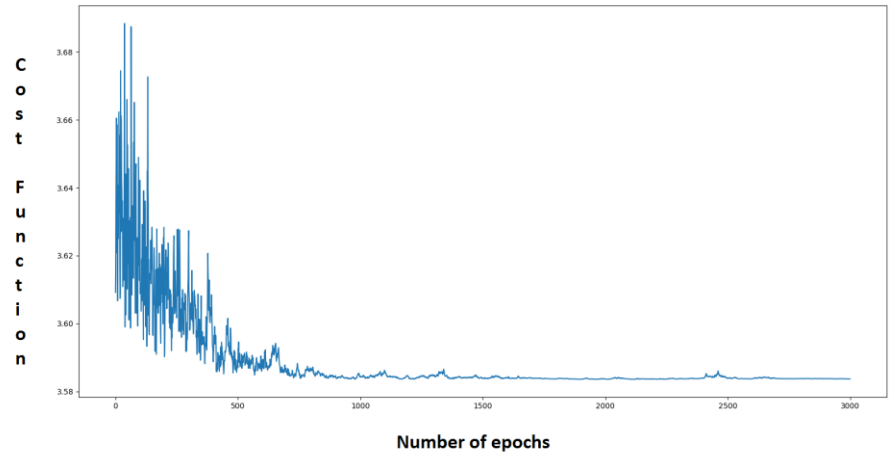

Figure 15: Cost function evaluated with the Telepresence dataset with the full network and a learning rate of 0.005

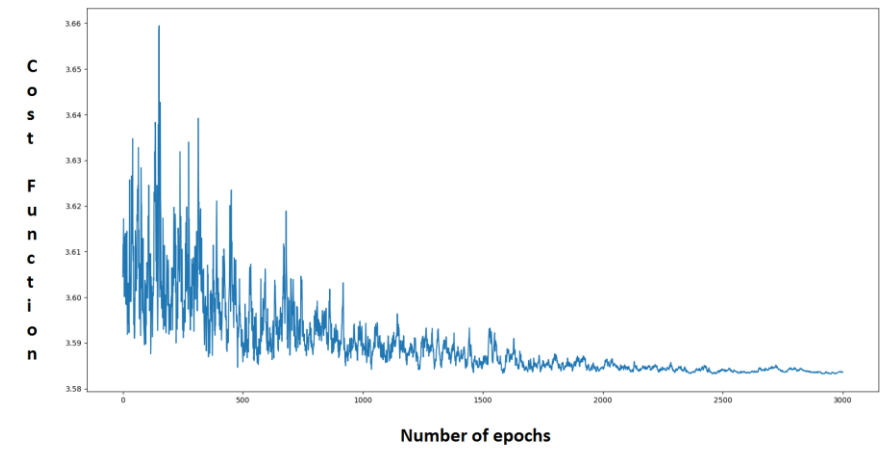

Figure 16: Cost function evaluated with the Telepresence dataset with the full network and a learning rate of 0.001

Another possible error I thought of was the possibility that the dataset had not both channels perfectly synchronized. To discard this idea, there were made two separated trainings using as input data just the Depth (*Figure 17*) or just the Color (*Figure 18*). As it can be appreciated, not only there is no noticeable difference between them but also both get stuck when reaching a contain point.

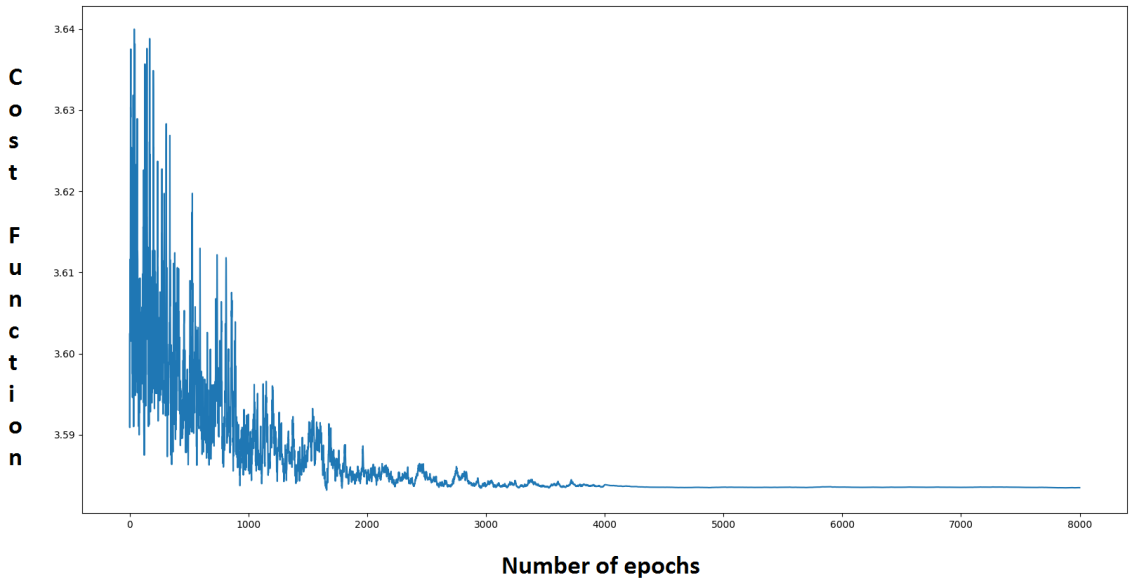

Figure 17: Cost function evaluated with the Telepresence dataset using only the Depth data

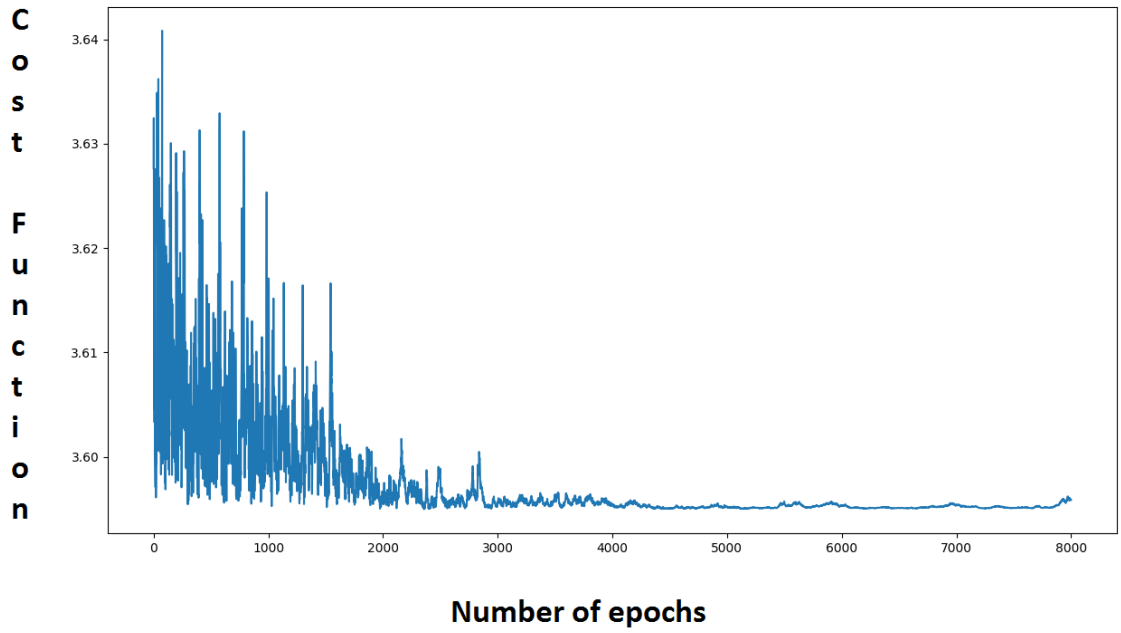

Figure 18: Cost function evaluated with the Telepresence dataset using only the Color data

The last thing tried, just to make sure that what was stuck was not the network itself, was selecting one random parameter from a random kernel of the network and plotted it through the full training. As it can be seen in *Figure 19*, even after the cost function had reached a saturation point, the selected parameter keeps changing, proving that the network is not stopped.

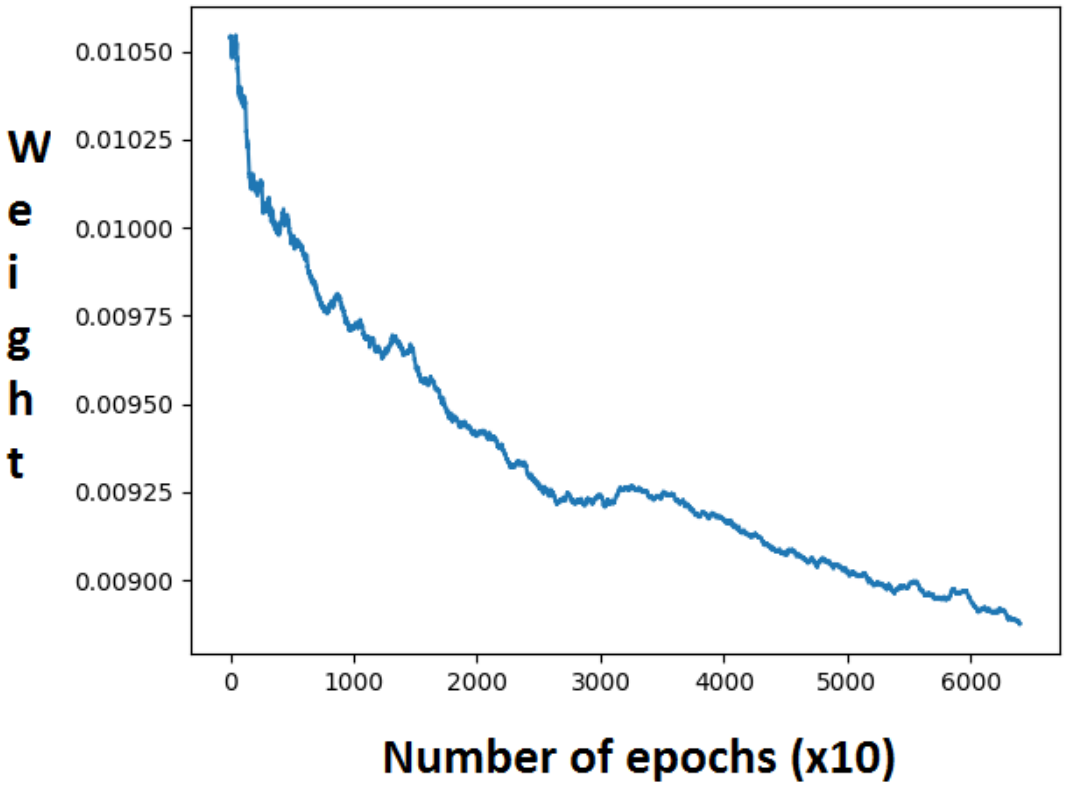

Figure 19: Value of a random weight through the training

Finally, having tried everything that came to my mind to see if the network was wrong and try to fix it, the only conclusion I could reach was that the Telepresence dataset had not been properly made. The main thing that comes to my mind is the fact that it was recorded as a single video and manually separated gesture by gesture. This process made them to be displaced through the temporal axis, which would be nice for a data augmentation technique but not for the basic training set, especially if it does not have plenty of elements.

Another reason that might have made the dataset not to be useful is the gesture speed, there are gestures performed by two different users where the first one makes it in 0.5 seconds and the second one in 2 seconds. This fact leads to the same result as the previous paragraph; A good technique for data augmentation but not for the basic set.

5. Budget

The budget of this project has been divided in two parts, the human costs and the hardware costs:

For the human costs, I have considered an average junior engineer salary of 10 €/h. Counting the hours invested in Python and TensorFlow training, the hours invested entirely in the project and the hours invested in recording and preparing the dataset they sum a total amount of 400 hours.

	Hours	Cost / Hour	Total Cost
Python/TensorFlow learning	80	10 €	800 €
Recording	20	10 €	200 €
Project	300	10 €	3000 €
Total	400	10 €	4000 €

Table 3: Human costs

For the hardware costs, I have just considered the cost of a single Kinect device v2 which is 89.91€ (99.99$) [11]

	Total Cost
Kinect v2	89.91 €

Table 4: Hardware costs

Due to the fact that the software used (TensorFlow) is free there is no need to add a software costs table.

	Total Cost
Human costs	4000 €
Hardware costs	89.91 €
Software costs	0 €
Total	4089.91 €

Table 5: Total costs

6. Conclusions and future development:

6.1. Conclusions

This project was initially born as a part of the Telepresence project, the idea was to integrate the gesture recognition for the system control. However, this has not been possible and the project has been carried only with “offline” data.

First of all, as an academical conclusion, I have to say that the major part of the initial objectives have been accomplished. Seeing the VIVA dataset results, we can assure that 3D-CNN are a really good deep learning technique when analyzing video sequences, and especially RGB-D video sequences. On the other hand, there was an initial objective that was not achieved; there was no way to train the network with the own recorded dataset. However, I do not want to see it only as a negative result; from it I learned many tips to try to fix a network before discarding a dataset

One thing I have to remark is that I have realized during this project that the dataset is far more important than what I thought when I learned the theoretical bases about it.

As a personal conclusion, I must say that doing this project has awoken in me a passion I did not know I have for Machine Learning which has led me to decide to do the Artificial Intelligence Master in the UPC.

6.2. Future Work

As a future work, I think there are two things that could be made to follow this project.

The first one would be recording a new dataset or, if I am wrong in my theory of why the Telepresence dataset has failed, find the real error with this one and if it can be fixed and work with it.

The second one would be to apply the trained model to real data. This might be done taking the input of a Kinect sensor with a window of 32 frames working as a FIFO (First In First Out) which, when a new frame enters, computes if there has been a gesture performed, and if so, do some action in consequence.

Bibliography:

[1] P. Molchanov, S. Gupta, K. Kim, J. Kautz. “Hand Gesture Recognition with 3D Convolutional Neural Networks”. CVPR 2015, Santa Clara, California, USA. Available: http://www.cv-foundation.org/openaccess/content_cvpr_workshops_2015/W15/papers/Molchanov_Hand_Gesture_Recognition_2015_CVPR_paper.pdf

[2] Codigofacilito. “Curso Python 3”. YouTube Playlist. 6 Sep 2016 - 18 Jan 2017. Available: https://www.youtube.com/playlist?list=PLpOqH6AE0tNiK7QN6AJo_3nVGQPc8nLdM

[3] Alcoverro, M. [et al.]. Gesture control interface for immersive panoramic displays. "Multimedia tools and applications", 25 Jul 2013. Available: http://upcommons.upc.edu/bitstream/handle/2117/20565/Gesture.pdf?sequence=1&isAllowed=y

[4] RedBit. “The evolution of interaction”. Microsoft Build 2016. Toronto. Available: http://www.redbitdev.com/microsoft-build-2016-toronto-hololens-for-business/

[5] Roberto Lopez. “Artificial Neural Network”. Neural Designer. Available: https://www.neuraldesigner.com/blog/perceptron-the-main-component-of-neural-networks

[6] Rob Hess, Clayton Mellina. “PARK or BIRD”. parkorbird. Available: http://code.flickr.net/2014/10/20/introducing-flickr-park-or-bird/

[7] Karpathy. “Convolutional Neural Networks (CNN/ Convnets)”. University of Stanford. Available: http://cs231n.github.io/convolutional-networks/

[8] D. Tran, L. Bourdev, R. Fergus, L. Torresani, M. Paluri. “Learning Spatiotemporal Features with 3D Convolutional Networks”. Darthmouth College. USA. Available: http://www.cv-foundation.org/openaccess/content_iccv_2015/papers/Tran_Learning_Spatiotemporal_Features_ICCV_2015_paper.pdf

[9] Laboratory for intelligent and safe automobile. “VIVA Hand Gesture Challenge”. Dataset. Available: http://cvrr.ucsd.edu/vivachallenge/index.php/hands/hand-gestures/

[10] NumPy Developers. “NumPy”. Python Library. Available: http://www.numpy.org/

[11] Microsoft. “Microsoft Kinect”. Official Website. Available: https://www.microsoft.com/en-us/store/d/kinect-sensor-for-xbox-one/91hq5578vksc

Glossary

- ANN: Artificial Neural Network
- CNN: Convolutional Neural Network
- 3D-CNN: 3D Convolutional Neural Network
- HRN: High Resolution Network
- LRN: Low Resolution Network
- ReLU: Rectified Linear Unit
- RGB: Red Green Blue
- RGB-D: Red Green Blue and Depth
- RNN: Recurrent Neural Network
- UPC: Universitat Politècnica de Catalunya (Polytechnic University of Catalonia)
- VIVA: Vision for Intelligent Vehicles and Applications