

# Identification and Monitoring of Violent Interactions in Video



Grado en Ingeniería Informática

## Trabajo Fin de Grado

Autor:  
Lourdes Mas Lillo

Tutor:  
Francisco Antonio Pujol López

Junio 2020



Universitat d'Alacant  
Universidad de Alicante



# Index

1	Tables .....	1
2	Figures.....	3
3	Special Thanks .....	9
4	Keywords .....	10
5	Abstract .....	10
6	Introduction .....	11
7	Motivation .....	11
8	Plan of Action .....	12
8.1	Goal and Objectives .....	13
8.2	Task Organization .....	14
8.2.1	Organization.....	15
8.2.2	System Preparation.....	15
8.2.3	Datasets .....	15
8.2.4	Training Comparative Studies.....	16
8.2.5	Alarm System.....	16
8.2.6	Project Documentation.....	17
9	State of the Art .....	17
9.1	Computer Vision Techniques.....	17
9.1.1	OViF.....	17
9.1.2	IFV .....	18
9.1.3	STIP.....	18
9.2	Neural Networks .....	18
9.2.1	DNN .....	20
9.2.2	CNN .....	20
9.2.3	RNN .....	21
9.2.4	LeNet5.....	24
9.2.5	AlexNet .....	24
9.2.6	MobileNet.....	25

9.2.7	SqueezeNet.....	25
9.2.8	VGG16 .....	25
9.2.9	LSTM .....	25
9.2.10	Stacked LSTM.....	26
9.2.11	Convolutional LSTM .....	27
9.3	Training Details.....	27
9.3.1	Validation Style.....	27
9.3.2	Optimizers .....	29
9.3.3	Activation Function.....	31
9.3.4	Loss Functions.....	37
9.3.5	Epochs .....	40
9.3.6	Batch Size.....	40
9.3.7	Time Steps.....	41
9.3.8	Training Host.....	41
9.3.9	Dropout.....	41
10	Methodology .....	42
10.1	Sensor Input Data.....	42
10.2	Datasets .....	42
10.3	Violence Detection.....	44
10.3.1	Computer Vision Techniques .....	45
10.3.2	Neural Networks.....	46
10.3.3	Learning Style .....	48
10.3.4	Pretraining .....	49
10.4	Tool Selection.....	49
10.4.1	Machine Learning Libraries .....	50
10.4.2	Language .....	50
10.5	Training Methodology.....	51
10.5.1	Normalization.....	51
10.5.2	Dataset.....	51

10.5.3	Choosing Optimizer .....	53
10.5.4	Choosing an Activation Function.....	54
10.5.5	Choosing a Loss Function .....	55
10.5.6	Interpreting the Accuracy and Loss.....	56
10.5.7	Choosing Number of Epochs, Batch Size and Time Steps .....	59
10.5.8	Choosing a Training Host .....	60
10.5.9	Computer Characteristics .....	60
10.5.10	Spatial and Computing Adaptations.....	61
10.6	Alarm System.....	62
10.7	Summary .....	63
11	System Description .....	63
11.1	Used Software .....	64
11.2	Training Comparative Studies.....	64
11.2.1	Models.....	64
11.2.2	ModelType Enumerator .....	68
11.2.3	Trainer .....	68
11.2.4	Custom data generators .....	71
11.2.5	Other Generators .....	73
11.2.6	Frames Provider .....	73
11.2.7	List ID Dictionary .....	74
11.2.8	Video Provider .....	75
11.3	Alarm System.....	76
12	Experimentation and Discussion.....	77
12.1	Training Comparative Studies.....	77
12.1.1	Datasets .....	78
12.1.2	Image Processing.....	82
12.1.3	Video Processing.....	83
12.1.4	Validations .....	97
12.1.5	Pretrain the Model.....	99

12.1.6	The Best Approximations.....	99
12.2	Alarm System.....	104
12.2.1	Optimization of the Alarm System.....	104
12.2.2	Timing.....	105
12.2.3	Accuracy Evaluation with a Custom Dataset.....	105
13	Review and Conclusion.....	106
13.1	Tasks Review.....	107
13.2	Project Review.....	107
13.3	Conclusion.....	109
14	Future Work.....	109
15	References.....	111
16	Annex.....	116
16.1	Gantt Diagram.....	116
16.2	Detailed Task Review.....	125
16.2.1	Organization.....	125
16.2.2	System Preparation.....	125
16.2.3	Datasets.....	125
16.2.4	Training Comparative Studies.....	126
16.2.5	Alarm System.....	126
16.2.6	Project Documentation.....	126
16.3	UML.....	128

# 1 Tables

Table 1: Logic steps to fulfil the objectives of this project, own elaboration ..... 14

Table 2: Indicators used to rate the value of a project task, own elaboration ..... 14

Table 3: Indicators used to rate the expected effort to achieve a project task, own elaboration ..... 14

Table 4: How to interpret the priority value of a task, own elaboration ..... 15

Table 5: Priority of tasks required for the organization of the project, own elaboration ..... 15

Table 6: Priority of tasks required for the system preparation of the project, own elaboration ..... 15

Table 7: Priority of tasks related with the datasets of the project, own elaboration..... 15

Table 8: Priority of tasks required for the training comparative studies of the project, own elaboration ..... 16

Table 9: Priority of tasks required for the alarm system of the project, own elaboration ..... 16

Table 10: Priority of tasks required for the documentation of the project, own elaboration..... 17

Table 11: Input and output size possible for a LSTM layer, own elaboration based on the LSTM structure..... 27

Table 12: Some examples of human action detection datasets and the references of their work, own elaborated list of remarkable datasets ..... 43

Table 13: INRIA IXMAS labels, own elaboration based on the IXMAS label content ..... 43

Table 14: UT-interaction labels, own elaboration based on the UT label content ..... 44

Table 15: Characteristics of the chosen datasets, own elaboration ..... 44

Table 16: Deep Learning frameworks of interest for the project and characteristics of them, own elaboration..... 50

Table 17: Characteristics of the possible programming languages to use in this project, own elaboration..... 50

Table 18: Analysis of the different styles to process the validation dataset, own elaboration ..... 53

Table 19: List of Keras optimizers with their main characteristics, own elaboration ..... 53

Table 20: Summary of the activation function characteristics, own elaboration ..... 54

Table 21: Summary of loss function characteristics, own elaboration..... 55

Table 22: Tools to interpret the results of the learning on a NN, own elaboration ..... 56

Table 23: Characteristics of the computer used during the training, own elaboration..... 60

Table 24: Options proposed to solve the lack of memory to load the dataset while training problem, own elaboration..... 62

Table 25: Layers size output of the LSTM NN, own elaboration ..... 67

Table 26: Layers size output of the ConvLSTM NN, own elaboration ..... 67

Table 27: Matrix of applied dataset modifications, own elaboration ..... 78

Table 28: Calculation of required training times with UT and IXMAS datasets, own elaboration . 78

Table 29: Results of the image-based models used in this project, own elaboration..... 83

Table 30: Accuracies obtained using data normalization with the IXMAS dataset, own elaboration .....	84
Table 31: Different accuracies obtained using data normalization with the IXMAS dataset, own elaboration.....	84
Table 32: Accuracies obtained using Adam instead of Adamax when the model converges, own elaboration.....	93
Table 33: Results during training of the models with the best outcome with an independent test, own elaboration.....	100
Table 34: The two best models obtained, and a given code to identify them, own elaboration.....	104
Table 35: Noise saved regarding violence in the frames for the LSTM model, own elaboration...	105
Table 36: Noise saved regarding violence in the frames for the ConvLSTM model, own elaboration .....	105
Table 37: System speed to detect violence in a bunch of 20 frames, own elaboration .....	105
Table 38: Testing used in the Alarm System, own elaboration.....	106
Table 39: Results of the test done on the alarm system for the best models created, own elaboration .....	106
Table 40: General assessment of the project tasks, own elaboration .....	107
Table 41: Priority of tasks required for the organization of the project, own elaboration .....	125
Table 42: Priority of tasks required for the system preparation of the project, own elaboration ...	125
Table 43: Assessment of tasks related with the datasets of the project, own elaboration .....	125
Table 44: Assessment of tasks required for the training comparative studies of the project, own elaboration.....	126
Table 45: Assessment of tasks required for the alarm system of the project, own elaboration.....	126
Table 46: Assessment of tasks required for the documentation of the project, own elaboration ...	126



## 2 Figures

Figure 1: PDCA cycle, the basic structure of the worldwide most applied quality standard EN ISO 9001, own elaboration based on (4) ..... 13

Figure 2: Deep learning from a macro perspective, own elaboration based on the artificial intelligence and deep learning dependency ..... 18

Figure 3: Unsupervised learning style, own elaboration ..... 19

Figure 4: Supervised learning style, own elaboration ..... 20

Figure 5: Reinforcement learning style, own elaboration ..... 20

Figure 6: Example of matrix multiplication on a convolution to calculate a pixel value applying a filter, own elaboration based on a filter application ..... 21

Figure 7: Deep CNN structure, own elaboration based in (10) ..... 21

Figure 8: Diagram with the RNN structure, own elaboration based on (10) ..... 22

Figure 9: Scheme of the fundamental RNN functionality, own elaboration based on (10) ..... 23

Figure 10: RNN structure in detail, own elaboration based on (10) ..... 23

Figure 11: The recursivity in a RNN diagram, own elaboration based on (12) ..... 24

Figure 12: Diagram visualizing simplified the LSTM structure, own elaboration based on (10)... 26

Figure 13: LSTM diagram: own elaboration based on the LSTM mechanism explanations found at (12) ..... 26

Figure 14: Cross-validation structure, own elaboration based on the cross-validation technique ... 28

Figure 15: One step ahead cross-validation structure, own elaboration based on the proposal at (20) ..... 29

Figure 16: Momentum and Nesterov behaviour, own elaboration based on (26) ..... 31

Figure 17: Identity function graphic representation, own elaboration using MATLAB ..... 32

Figure 18: Binary step, own elaboration using MATLAB ..... 32

Figure 19: Sigmoid function graphic representation, own elaboration using MATLAB ..... 33

Figure 20: tanh function graphic representation, own elaboration using MATLAB ..... 33

Figure 21: arctan function graphic representation, own elaboration using MATLAB ..... 34

Figure 22: ReLU function graphic representation, own elaboration using MATLAB ..... 35

Figure 23: Leaky ReLU function graphic representation, own elaboration using MATLAB ..... 35

Figure 24: Parametric ReLU function graphic representation, own elaboration using MATLAB .. 36

Figure 25: ELU graphic representation, own elaboration using MATLAB ..... 36

Figure 26: ReLU-6 function graphic representation, own elaboration using MATLAB ..... 37

Figure 27: Softmax function graphic representation, own elaboration using MATLAB ..... 37

Figure 28: Frames from the IXMAS dataset that belong to a kick labelled video ..... 45

Figure 29: Human shape, own elaboration using as background a frame from the UT dataset ..... 47

Figure 30: A video frame shown in RGB colour and in greyscale, taken from the UT dataset and edited..... 52

Figure 31: Accuracy evolution of a non-learning model, created using matplotlib ..... 57

Figure 32: Loss evolution of a non-learning model, created using matplotlib..... 57

Figure 33: Accuracy of a validation set unrepresentative during the training of a ConvLSTM NN, created using matplotlib during the tests made in this project ..... 59

Figure 34: Loss of a validation set unrepresentative during the training of a ConvLSTM NN, created using matplotlib during the tests made in this project..... 59

Figure 35: Behaviour of the alarm system, own elaboration..... 63

Figure 36: The stacked LSTM model network structure created, own elaboration ..... 65

Figure 37: The ConvLSTM model network structure created, own elaboration..... 67

Figure 38: Screenshot that shows the indexes that point to the batches made for the training, taken during a program run..... 72

Figure 39: Console application menu for executing the alarm system, screenshot of the program run ..... 76

Figure 40: Alarm System running during a violent situation, screenshot of the program run ..... 76

Figure 41: Alarm System running after a violent situation, screenshot of the program run ..... 77

Figure 42: Screenshot of frames from a UT dataset video, extracted from the UT dataset ..... 79

Figure 43: Screenshot of frames from the UT dataset that show that there are video frames where there is not the action labelled happening, approximately the first 25 frames of this video express no movement, extracted from the UT dataset ..... 80

Figure 44: Screenshot of frames from the UT dataset that show that the frames placed on the middle of a video express the labelled action, extracted from the UT dataset ..... 80

Figure 45: Model accuracy graphic for LSTM with binarized IXMAS dataset, created using matplotlib during the tests made in this project..... 81

Figure 46: Model loss graphic for LSTM with binarized IXMAS dataset, created using matplotlib during the tests made in this project..... 81

Figure 47: Results obtained training a model with a binarized dataset, screenshot from a program run ..... 82

Figure 48: Mistaken frames while binarization from the IXMAS dataset, extracted from the IXMAS dataset..... 82

Figure 49: Stacked LSTM model accuracy with MSE loss and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project..... 85

Figure 50: Stacked LSTM model loss with MSE loss and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project..... 85

Figure 51: Stacked LSTM model accuracy for 20 time steps and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project..... 86

Figure 52: Stacked LSTM model loss for 20 time steps and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project ..... 87

Figure 53: Stacked LSTM model accuracy for 10 time steps, batch size of 20 and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project ..... 88

Figure 54: Stacked LSTM model loss for 10 time steps, batch size of 20 and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project ..... 88

Figure 55: Stacked LSTM model accuracy for 10 time steps, batch size of 80 and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project ..... 89

Figure 56: Stacked LSTM model loss for 10 time steps, batch size of 80 and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project ..... 89

Figure 57: Screenshot that shows the predictions for each possible label for a frame from the UT dataset, screenshot from a program run ..... 90

Figure 58: Model accuracy graphic for leaky ReLU model using an alpha of 0.01 with Adamax and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project ..... 91

Figure 59: Model loss graphic for leaky ReLU model using an alpha of 0.01 with Adamax and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project ..... 91

Figure 60: Model accuracy graphic for leaky ReLU model using an alpha of 0.05 with Adamax and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project ..... 92

Figure 61: Model loss graphic for leaky ReLU model using an alpha of 0.05 with Adamax and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project ..... 92

Figure 62: Model accuracy graphic for leaky ReLU model using an alpha of 0.05 with Adam and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project ..... 94

Figure 63: Model loss graphic for leaky ReLU model using an alpha of 0.05 with Adam and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project ..... 94

Figure 64: Stacked LSTM model accuracy with five layers of deepness and sigmoid final layer activation, created using matplotlib during the tests made in this project..... 95

Figure 65: Stacked LSTM model loss with five layers of deepness and sigmoid final layer activation, created using matplotlib during the tests made in this project ..... 95

Figure 66: Stacked LSTM model accuracy with three layers of deepness and softmax final layer activation, created using matplotlib during the tests made in this project..... 96

Figure 67: Stacked LSTM model loss with three layers of deepness and softmax final layer activation, created using matplotlib during the tests made in this project ..... 96

Figure 68: Accuracy in the last fold of a cross-validation using the complete IXMAS dataset and the ConvLSTM model, created using matplotlib during the tests made in this project ..... 98

Figure 69: Loss in the last fold of a cross-validation using the complete IXMAS dataset and the ConvLSTM model, created using matplotlib during the tests made in this project ..... 99

Figure 70: Stacked LSTM model accuracy for 10 time steps and a learning rate of $1e^{-3}$ , created using matplotlib during the tests made in this project.....	100
Figure 71: Stacked LSTM model loss for 10 time steps and a learning rate of $1e^{-3}$ , created using matplotlib during the tests made in this project.....	101
Figure 72: ConvLSTM model accuracy with 0.5 dropout, created using matplotlib during the tests made in this project .....	102
Figure 73: ConvLSTM model loss with 0.5 dropout, created using matplotlib during the tests made in this project.....	102
Figure 74: ConvLSTM model accuracy with 0.75 dropout, created using matplotlib during the tests made in this project .....	103
Figure 75: ConvLSTM model loss with 0.75 dropout, created using matplotlib during the tests made in this project.....	103
Figure 76: Legend of the color code used in the Gantt Diagram, own elaboration.....	116
Figure 77: Gantt diagram for the month of October 2019, own elaboration.....	117
Figure 78: Gantt diagram for the month of November 2019, own elaboration.....	118
Figure 79: Gantt diagram for the month of December 2019, own elaboration .....	119
Figure 80: Gantt diagram for the month of January 2020, own elaboration .....	120
Figure 81: Gantt diagram for the month of February 2020, own elaboration .....	121
Figure 82: Gantt diagram for the month of March 2020, own elaboration .....	122
Figure 83: Gantt diagram for the month of April 2020, own elaboration .....	123
Figure 84: Gantt diagram for the month of May 2020, own elaboration .....	124
Figure 85: Trainers UML, own elaboration .....	128
Figure 86: VideoProvider UML, own elaboration .....	129
Figure 87_Alarm System UML, own elaboration.....	130
Figure 88: Menu UML, own elaboration .....	130
Figure 89: Other generators UML, own elaboration.....	131

“That is what learning is. You suddenly understand something you’ve understood all your life, but in a new way.”

Doris Lessing



### 3 Special Thanks

The final project is a significant task where the capabilities of the student are tested at the end of the degree. It is also a very important moment of life, each step on its creation brings the student closer to a big change on their life... what comes after the studies are finished? I am taking these steps full of joy, motivation and thankfulness to all the people that accompany me and support me all this way.

I would like to take this opportunity to thank you, mom and dad, because, since I was a kid, you taught me to be perseverant and care about my future. Thanks to you, I learned to reduce leisure time when I needed to focus on my studies but still always have time for a good paella break on Sunday. It is your tips that brought me to where I am. In addition, of course, thanks for having me home these extra years and for your support. I want to specially mention you, dad, for being the first believing in me when I decided to go back to university and study a completely new field for me, computing.

This might be an irony of the destiny, Chris, as my best friend, you were in Dublin the first one to know about my plans to go back to university. Now, as my husband, in Bamberg you sit patiently by my side while I finish this writing, being the first one reading the documentation that portray these years of work.

Also, to you, Paco, my tutor. I had the opportunity to be student of one of your subjects before, were you shared dynamism. We were programming robots to follow fun circuits but also having interesting lectures, debates, opinion pieces..., you showed us how to do a good job and enjoy it. It is a pleasure to work again with you for this final project, even though being in different countries and not being able to do the presentation physically due to COVID-19 restrictions, it is an enriching experience to share the progress of this project over the screen. Your dedication is an example that I will keep on for my future steps.

I am a lucky person to have people in my life that believe in me, support me and teach me. I have enjoyed the years at university, this final project and I am feeling prepared to continue my path with perseveration, patience and dedication.

## 4 Keywords

Alarm system, AlexNet, bullying, ConvLSTM, convolutional long short-term memory, convolutional neural network, dataset, graphic processing unit, Keras, LeNet5, long sort-term memory, LSTM, normalization, NN, neural network, NumPy, OpenCV, PDCA, pretraining, recurrent neural network, RNN, surveillance camera, TensorFlow, time steps, video processing, video recognition, violence, violence detection.

## 5 Abstract

This project shall help to bring a tool to fight against bullying in schools. It is also possible to use it in different scenes where a camera is recording a common area shared by people, such as companies, banks, prisons, or hospitals. To achieve that, the issue is approached from two main modules. The first one, a comparative study of approaches to detect violence in video, using image and video analyser Neural Networks (NN)s: a custom image analyser NN based on LeNet5, AlexNet, custom stacked long short-term memory (LSTM) and convolutional LSTM based NNs. The trainings are done with two datasets that have been subject to modifications to correct possible misinterpretations during the learning and pretraining is applied. The LeNet5 based NN is unsuccessful and tested with an independent dataset AlexNet is inaccurate. The best results are obtained with a stacked LSTM NN and a convolutional LSTM with dropout and a LSTM layer. Both NNs achieve over 90 % of accuracy with training and validation datasets, meanwhile the stacked LSTM and the convolutional NN achieve, respectively, 75 % and 100 % of accuracy with a small independent test dataset created. The convolutional LSTM needed 10 times less epochs to achieve the same result as the stacked LSTM.

The second module consists of a violence detection system that applies the best solution obtained from the comparative study. The violence detection system saves the frames detected as violence with date, time and camera name and emits a sound alarm when more than a certain number of consecutive frames are evaluated as containing violence. This way the sensitivity of the system is reduced and avoids false alarms due to small mistakes done by the intelligence.



## 6 Introduction

As the Institute of Statistics of the UNESCO released, almost a third of the teenagers in the world have suffered bullying recently (1). Some figures are incredibly high, especially in developing countries. The percentages are not small in Europe either, for example, 23.27 % in Germany or 39.04 % in Portugal, Spain was at 15.37 %. These data belong to the result of the statistics obtained in 2014 for the Sustainable Development Goal and represent the percentage of students experiencing bullying in the last 12 months. Furthermore, it should be considered that these statistics only contain registered cases.

Esteban Beltrán, director of Amnesty International Spain, talks in (2) about the lack of registration regarding bullying cases in Spanish schools. He explains difficulties to identify such cases and that the current measures to handle them are not working sufficiently. In the same report Beltrán adds that the protection of the infancy is an obligation and therefore, not fulfilling this duty is a violation of the human rights.

## 7 Motivation

As a Social Worker finishing my degree in Computer Engineering, I have proposed myself to merge my two fields of expertise in this final degree project: A computing project with the aim to promote the social development and help to increase the wellness of people. Part of our duty as a society is to keep order to maintain a good coexistence of everyone regardless differences between the individuals. This means giving everyone the prerequisites to develop their life in a safe way fulfilling their basic needs. The Universal Human Rights Declaration, published by the United Nations declares the basic rights that apply to all humanity with the intention to recognize their dignity freedom and justice rights (3).

Aggression is part of human history and still is, from macro events as the fall of the Roman Empire or the World Wars, to smaller happenings like disputes for access to resources or bullying at schools. At violence situations, where one or several individuals are attacking a third individual or another group, some of the articles of the Universal Human Rights Declaration are broken which means committing a big offense against the basic wellness needs of the offended person. Violence acts against a person disrespect the following articles of the Human Rights Declaration:

First place, article number 3, because offended persons are forced to a change of their situation that is not wanted and therefore, they lose their liberty and security.

*“Article 3. Everyone has the right to life, liberty and security of person.” (3)*

The physical and mental wellness of a person is also attacked if not following the 5<sup>th</sup> article, due to a violence act against them.

*“Article 5. No one shall be subjected to torture or to cruel, inhuman or degrading treatment or punishment.” (3)*

The honour of the person under aggression is insulted because its bounds to society are damaged by disrespectful treatment. This is breaking the rights given to an individual by the article number 12 of the currently analysed declaration.

*“Article 12. No one shall be subjected to arbitrary interference with his privacy, family, home or correspondence, nor to attacks upon his honour and reputation. Everyone has the right to the protection of the law against such interference or attacks.” (3)*

Social work is a discipline that promotes the social change, solving problems in human relationships and works for the empowerment and freedom of the individual to increment their wellness. Therefore, social work intervenes on the interaction of the people with their environment.

Personally, from the social worker point of view, I feel the need to act for a better society that secures the basic rights of all individuals, especially for those individuals more vulnerable. After evaluating potential projects to finalize my degree as Computer Engineer, I found a possibility to combine learned abilities to fulfil what my moral considers correct. Creating this tool may help the society by detecting violence in institutions, which increases the wellness of individuals that suffer from bullying.

Intervention against violence in institutions should be confronted taking various actions: In one hand, it is important to work on the negotiation capability and empathy of all the members of the institution. This step can be defined as prevention. In the other hand it is important to detect current violence situations to be able to interfere and re-educate the aggressor and support victims giving them protection and methods to confront this situation and possible similar ones in future. A fast detection is key to de-escalate or eliminate an issue and make it less natural and accepted as something common. Moreover, the violence will have a reduced impact time onto the individuals involved making it easier to be treated. Analysing this information, emerges the need to improve the possible detection time of violence by bringing a tool that can help to stop bullying situations, in a fast way.

## 8 Plan of Action

This chapter explains the organization followed in this project. First, the goal and the objectives are set, and after, the tasks required to fulfil the objectives are worked out and prioritized. To ensure a high-quality result, this project is organised following the basic structure of the world's most applied quality management standard EN ISO 9001 (4). This standard is based on the idea of continuous improvement to optimize the output of any project or process. The PDCA cycle shown in Figure 1 explains the main steps to achieve continuous improvement and the closed control circuit of this method ensures the high-quality output.

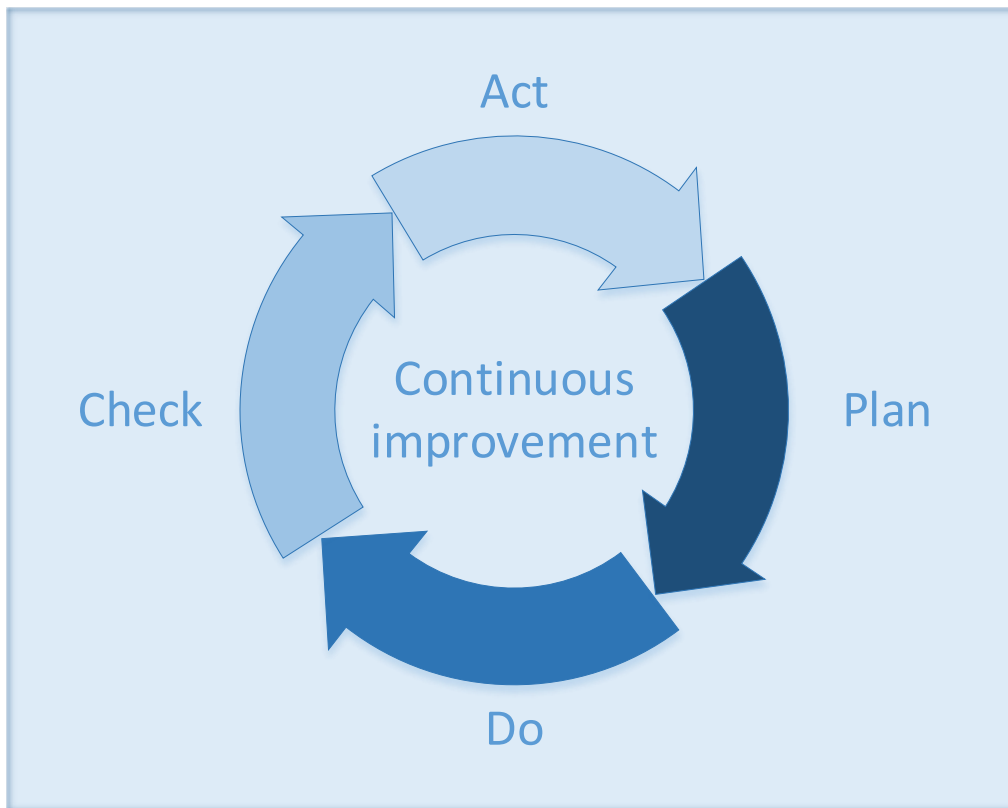


Figure 1: PDCA cycle, the basic structure of the worldwide most applied quality standard EN ISO 9001, own elaboration based on (4)

The PDCA cycle is mirrored in the general structure implemented in this project:

- Plan (Chapters 8, 9 and 10)
- Do (Chapters 11 and 12)
- Check (Chapter 13)
- Act (Chapter 14)

Also, within the experimental section the control circuit ensures continuous improvement due to each result getting analysed and the gained know-how being implemented into the plan of the following experiment.

## 8.1 Goal and Objectives

To accomplish the goal developed in the motivation (chapter 7), “to improve the possible detection time of violence by bringing a tool that can help to stop bullying situations, in a fast way”, there are some steps that are considered required. In first place, it is necessary to fast and accurately detect aggression acts while being executed. For conventional methods are considered too slow, automatization could solve this problem. Second, it is required to produce evidence of the aggression that could be used for analysis and determine the gravity. And in third place, it is required that the competent authority gets informed quickly to interfere and stop the aggression. From these premises three objectives are extracted that will ensure the goal of this project:

- I. Detecting danger situations in video in an automatized way
- II. Saving the information related with the possible aggression
- III. Generating an alarm to warn the competent authority

To fulfil the objectives, two technical modules can be distinguished, a first one that fast and automatized detects violence scenes as required for the first objective. And a second module, a tool to record the detected scenes and generate an alarm, accomplishing the second and third objective. To simplify them, Table 1 shows the names given to the modules and their logic order of processing, these names are used through the document to organize the explanations.

Table 1: Logic steps to fulfil the objectives of this project, own elaboration

Module	
<b>First module</b>	Training comparative studies
<b>Second module</b>	Alarm system

## 8.2 Task Organization

In this section, tasks necessary to fulfil the goal and objectives are worked out and organized. In addition, the tasks have been rated according to their contribution to the project result to finally enable proper prioritization and organisation of resources while project controlling. Table 2 and Table 3 show the explanation how the severity and effort of a task are estimated. Noticeably the severity value increases by 5 for each step and therefore is favoured over the effort. The reason is that the most crucial tasks must be done to achieve full functionality, even if their effort is high.

Table 2: Indicators used to rate the value of a project task, own elaboration

Value for the project output (VAL)	
Indicator	Meaning
<b>5</b>	Very low severity, this task has a cosmetic impact on the project result
<b>10</b>	Low severity, this task has a low impact on the project result
<b>15</b>	Medium severity, this task has a medium impact on the on the project result
<b>20</b>	High severity, this task is mandatory for the system to work properly and be useful

Table 3: Indicators used to rate the expected effort to achieve a project task, own elaboration

Estimated effort (EE)		
Indicator	Meaning	Expected time effort [t]
<b>6</b>	Very low effort	$t < 1$ day
<b>5</b>	Low effort	$1 \text{ day} \leq t < 2 \text{ days}$
<b>4</b>	Medium -low effort	$2 \text{ days} \leq t < 1 \text{ week}$
<b>3</b>	Medium-high effort	$1 \text{ week} \leq t < 2 \text{ weeks}$
<b>2</b>	High effort	$2 \text{ weeks} \leq t < 1 \text{ month}$
<b>1</b>	Very high effort	$t \geq 1 \text{ month}$

Finally, the priority of a tasks is calculated by multiplying the severity and effort values. Table 4 shows how to interpret the results of this calculation and their meaning regarding their importance

for the project efficiency. The calculation results per task are organized and presented in the Table 5, Table 6, Table 7, Table 8, Table 9, Table 10

Table 4: How to interpret the priority value of a task, own elaboration

Priority (PRIO)	Description
$PRIO \leq 30$	Indifferent for the project
$60 \geq PRIO > 30$	Recommended task
$PRIO > 60$	Necessary task

### 8.2.1 Organization

Table 5: Priority of tasks required for the organization of the project, own elaboration

1. Organization					
Code	Task	Details	VAL	EE	PRIO
1.1	Plan	Produce the list of tasks determinant to fulfil the objectives of the project	20	5	100
		Organize the tasks depending on their importance for the project	20	6	120
		Create a time plan	15	5	75
1.2	Review Progress	Summarise the status of the project	15	6	90
		Check if the progress reached fulfils the plan	15	6	90
1.3	Reunion with tutor	Explain progress to the tutor	15	6	90
		Explain important changes to the tutor	15	6	90
		Discuss the next steps with the tutor	15	6	90

### 8.2.2 System Preparation

Table 6: Priority of tasks required for the system preparation of the project, own elaboration

2. System preparation					
Code	Task	Details	VAL	EE	PRIO
2.1	Research options	Choose the favourite options for building and running the project	20	5	100
2.2	Installations	Install required software	20	5	100
		Make the required installations for the dependency with libraries and frameworks	20	5	100
2.3	Ensure protection to data	Apply a license to the project	5	6	30
		Use a control version	20	6	120

### 8.2.3 Datasets

Table 7: Priority of tasks related with the datasets of the project, own elaboration

3. Datasets					
Code	Task	Details	VAL	EE	PRIO
3.1	Dataset research	Acquire datasets that are of use for the project	20	4	80
		Analyse scholar literature (journal articles, conference proceedings, books, ...) related to works where those datasets were used	20	4	80

3.2	Structure data	Design suitable datasets for the model from the original video datasets to use during the training	20	5	100
3.3	Implement code to handle input	Create code to automatically modify big datasets, reorganize and relabel	20	4	80

### 8.2.4 Training Comparative Studies

Table 8: Priority of tasks required for the training comparative studies of the project, own elaboration

4. Training comparative studies					
Code	Task	Details	VAL	EE	PRIO
4.1	Research	Analyse the related theory using literature (journal articles, conference proceedings, books, ...)	15	4	60
		Analyse the current state of the art using literature (journal articles, conference proceedings, books, ...)	15	4	60
		Compare different model approximations	15	4	60
4.2	Design and implement	Apply learned information to design an own NN	20	4	80
		Implement the NN	20	5	100
4.3	Train single NN	Experiment with different NN	15	3	45
		Execute trainings of the NN with different datasets	15	3	45
4.4	Use with pretrained NN	Execute retraining of a model with different datasets	15	3	45
4.5	Modify and update NN	Modify NN to improve the results based on the evaluation	20	6	120
4.6	Compare with previous results	Analyse the results obtained from the trainings	10	6	60
		Compare the accuracy of different datasets	10	6	60
		Analyse how different values affect the results of the training	15	6	90

### 8.2.5 Alarm System

Table 9: Priority of tasks required for the alarm system of the project, own elaboration

5. Alarm system					
Code	Task	Details	VAL	EE	PRIO
5.1	Create automatized detector	Use trained models to evaluate violence detector in video	20	5	100
		Create interface	5	5	25
		Frame the people on the scenes	5	5	25
		Store the detected violence video sections	15	6	90
		Signalize if a violence situation is detected by the system	20	6	120
5.2	Test	Test the alarm with an independent dataset	15	5	75

### 8.2.6 Project Documentation

Table 10: Priority of tasks required for the documentation of the project, own elaboration

6. Project documentation					
Code	Task	Details	VAL	EE	PRIO
6.1	Create structure	Customize the documentation layout to fulfil the layout of the University of Alicante requirements for final projects	20	6	120
6.2	Introduction	Generate project introduction documentation	20	4	80
6.3	Theoretical knowledge	Generate documentation of the studies obtained related to the project	20	2	40
		Organize the bibliography into a folder	5	6	30
6.4	Implementation and testing	Generating documentation of the programming progress	20	3	60
		Generating documentation of the tests realized	20	3	60
		Generating documentation of the result and interpretation	20	4	80
6.5	Summary and evaluation	Evaluating the results of the training comparative studies	20	5	100
		Evaluating the results of the alarm system	20	5	100
		Assessment of the project objectives	15	6	90
		Conclusion and future work	20	6	120
6.6	Revise documentation	Correct mistakes in the documentation	20	4	80

## 9 State of the Art

This chapter introduces concepts that support understanding the general operating principle and modules needed to develop a system able to fulfil the objectives. In first place, a selection of computer vision techniques used to detect human actions in video and in second place, Neural Networks (NN)s that learn from video data are defined.

### 9.1 Computer Vision Techniques

The input data for this system should be properly prepared to enable it to recognize and process visual data efficiently with NNs. In this chapter, techniques to process video and image data are introduced. Those can help by focussing specific features that are fed to the NN.

#### 9.1.1 OViF

Oriented Violent Flows (OViF) is a method proposed for fast practical detection of violence in videos. It is taking advantage of the motion orientation of data regarding the magnitude of the change of information. Hence, this method helps focussing on parts of the input data that shows a rapid change in its consistency, like for example a fast moving leg performing a kick. (5)

### 9.1.2 IFV

The Improved Fisher Vectors (IFV) encoding simplifies the structure of videos by connecting the space-time of the referent points' positions in the image. This has use, for example, to determine their speed or trajectory. (6)

### 9.1.3 STIP

For the Space Time Interest Point (STIP) the variation in space and time is analysed to find the interest points. That have the highest amount of displacements on the video. For a video showing a fox moving in front of a static plant for example, the STIP is formed around the animal. Through a segment of 2-dimensional images, the method marks interest points that evolve through the frames in 3-dimensions, the position (x,y) dependent on the time. (7) (8)

## 9.2 Neural Networks

Artificial Intelligence (AI) is the umbrella term for intelligence associated with machines that are capable to solve a problem independently of the method. A logic algorithm or Machine Learning (ML) can achieve this, for example. ML is a branch of the AI that is focused on the use of learning from generalizations to solve a problem. An evolution of the ML is the Deep Learning (DL), where NNs are applied to imitate the functionality of the visual cortex of animals. This technique is very successful detecting image and audio characteristics (9). In Figure 2 is presented the dependency order between the three elements just explained. The following chapter defines the basic functionality of a NN, introduces a selected list of specialised NNs architectures and finally explains settings necessary for training of a NN.

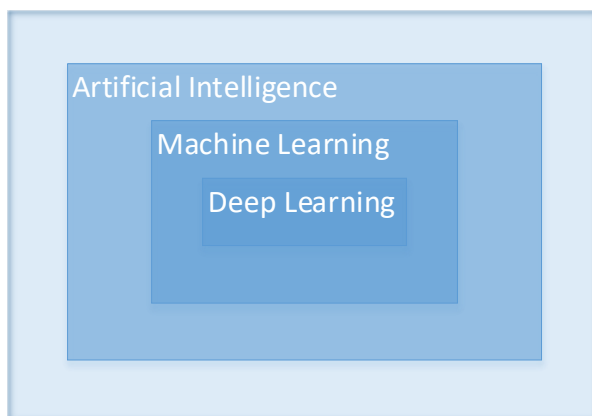


Figure 2: Deep learning from a macro perspective, own elaboration based on the artificial intelligence and deep learning dependency

### Neuron

Neurons are the components of the NNs. They are interconnected and show a similar behaviour as biological neurons processing and passing data for interpretation. Each neuron is a mathematical function that computes the weight average of the input it receives, and then applies an activation function to generate an output for the next neuron.



**Network Structure**

A NN is typically organized in an input layer, hidden layers and an output layer and each neuron is connected to the neurons set on the previous layer. It basically consists in a circuit of neurons, which connections are set as weights. Those weights can be of positive or negative value and, if positive, promote the connection, while, if negative, inhibits it. The inputs, modified by the weights are summed on their way through the net and, finally, applying an activation function determines the output range. This method makes it possible to find patterns by building complex relationships between the data input and the output.

**Activation Functions**

The activation function is an important part of a training process of a NN. This function determines, depending on the importance of the input, how much the weight of a neuron is transformed. Therefore, it will strongly influence the output of a NN and is determinant for a correct learning process. Further details to activation functions and how to use them can be found in chapter 9.3.3.

**Learning Styles**

Various learning styles can be applied to NNs as, in first place, the Unsupervised Learning (UL). With this method, unlabelled data is analysed to find common patrons and create groups of data an scheme of this learning is presented in Figure 3. In second place, the Supervised Learning (SL) learns from data that contain labels. During the training the system tries to predict the label of the elements and learns from comparing with the real result as seen at Figure 4. Last, is the Reinforcement Learning (RL), this method represented at Figure 5, is an interactive system based on learning from feedback of the environment.

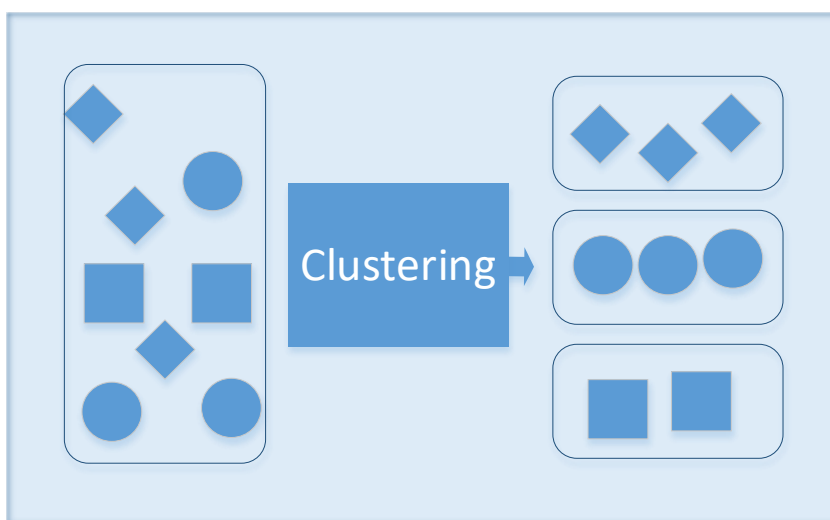


Figure 3: Unsupervised learning style, own elaboration

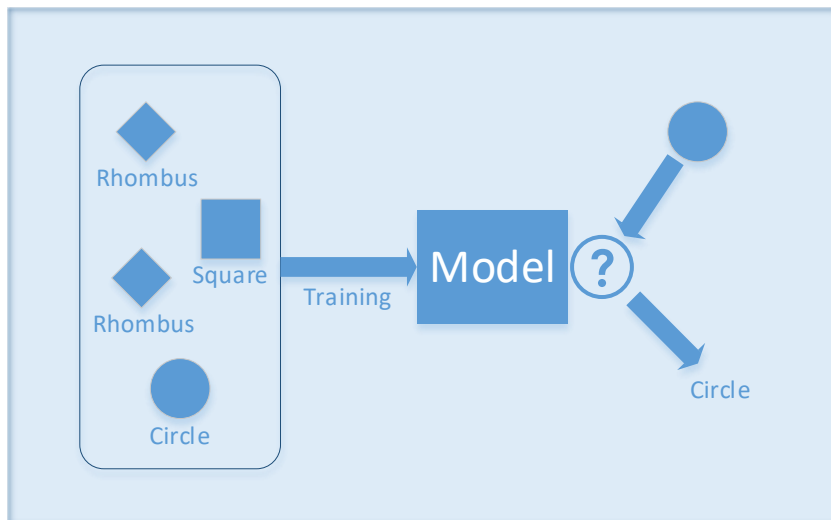


Figure 4: Supervised learning style, own elaboration

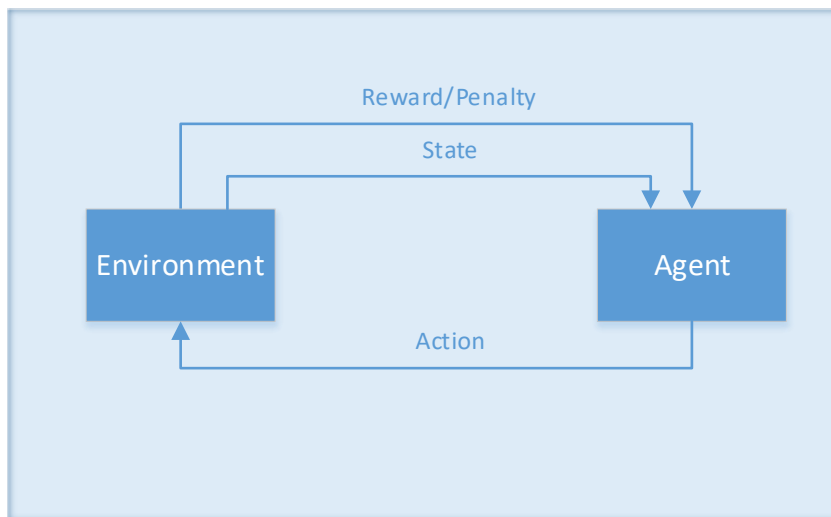


Figure 5: Reinforcement learning style, own elaboration

### 9.2.1 DNN

A NN is divided in layers containing each a certain quantity of neurons. A simple NN contains three layers, one for the input, a hidden layer, and another for the output. A Deep Neural Network (DNN) consists on a NN with more than one hidden layer. There can be the same quantity of neurons in two NNs with a different quantity of layers, for example, having one layer with ten neurons or ten layers with one neuron each. The layers are used to find features with different level of abstraction, from more general to more detailed. For example, for learning how to recognize humans in images, a first layer can determine general curvatures, a second one shapes of bodies, and a third one parts of the bodies. Therefore, the use of DNN is of help processing complex problems.

### 9.2.2 CNN

This NN extracts features from the data using a kernel which is a matrix operation applied across the data as shown in Figure 6. It is characterized by having multilinear operations that happen in the hidden layers. The basic operations of the CNN are the convolutions and the pooling. The first one

consists of applying filters to find features and the second consists of subsampling to reduce the dimensions of the feature map. A simple scheme of a CNN is shown in Figure 7 for a better understanding. The CNNs are challenging the state of the art in activity recognition (9).

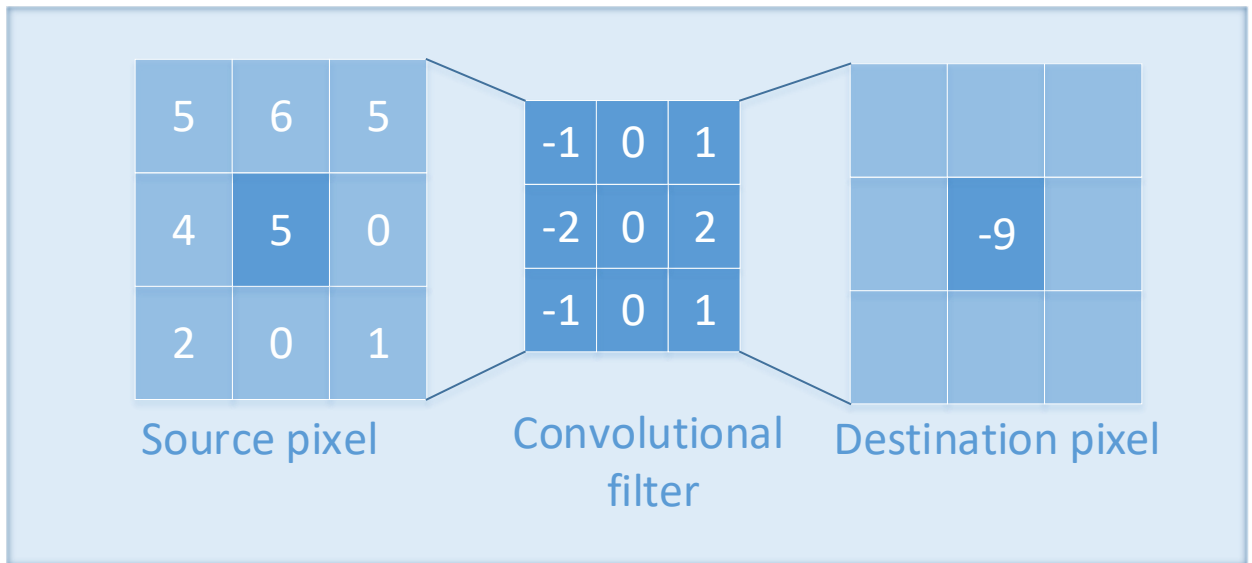


Figure 6: Example of matrix multiplication on a convolution to calculate a pixel value applying a filter, own elaboration based on a filter application

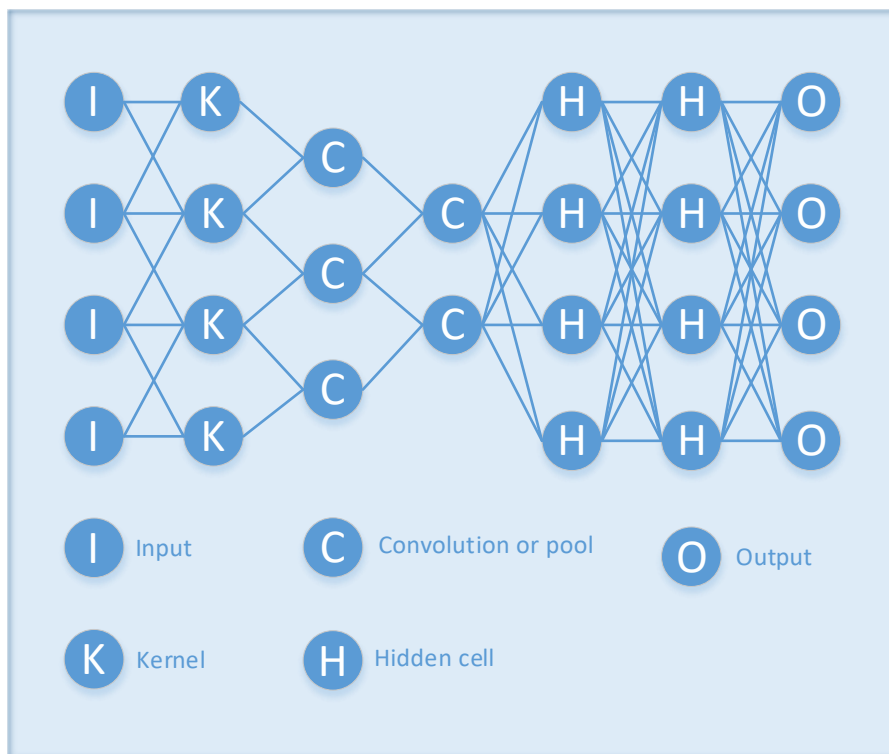


Figure 7: Deep CNN structure, own elaboration based in (10)

### 9.2.3 RNN

The idea of the Recurrent Neural Network (RNN) started when the back-propagation algorithm was developed. The fundament for an RNN is the recursion. It is a powerful procedure applied to obtain a complex function with simple steps through repetition. It enables to fine tune the weights of its

neurons on base of the error rate. Applying recursion to artificial intelligence can bring particularly good results. (11)

The structure of an RNN, as represented in Figure 8, consists of keeping the input and its context unities with the objective of maintaining the information from the past, while looping with the purpose of updating the status in the RNN.

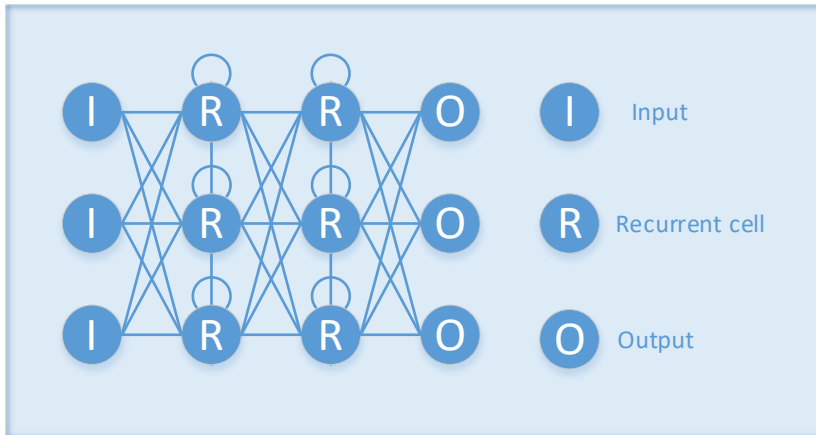


Figure 8: Diagram with the RNN structure, own elaboration based on (10)

The general recursive formula applied for the RNN is shown at [1]:

$$[1] \quad S_t = F_w(S_{t-1}, X_t)$$

Where  $S_t$  is the state at the time step  $t$ ,  $F_w$  is a recursive activation function dependent on  $S_{t-1}$  and  $X_t$ .  $S_{t-1}$  represents the state at the previous time step and  $X_t$  is the input at time step  $t$ . To formulate an example, in [2] is assumed that the activation function  $F_w$  is a tanh function.

$$[2] \quad S_t = \tanh(W_s S_{t-1} + W_x X_t)$$

The state at the time step  $t$  is the update of the weighted previous state  $W_s S_{t-1}$  with the current weight per the current input  $W_x X_t$  and passed through a tanh activation.

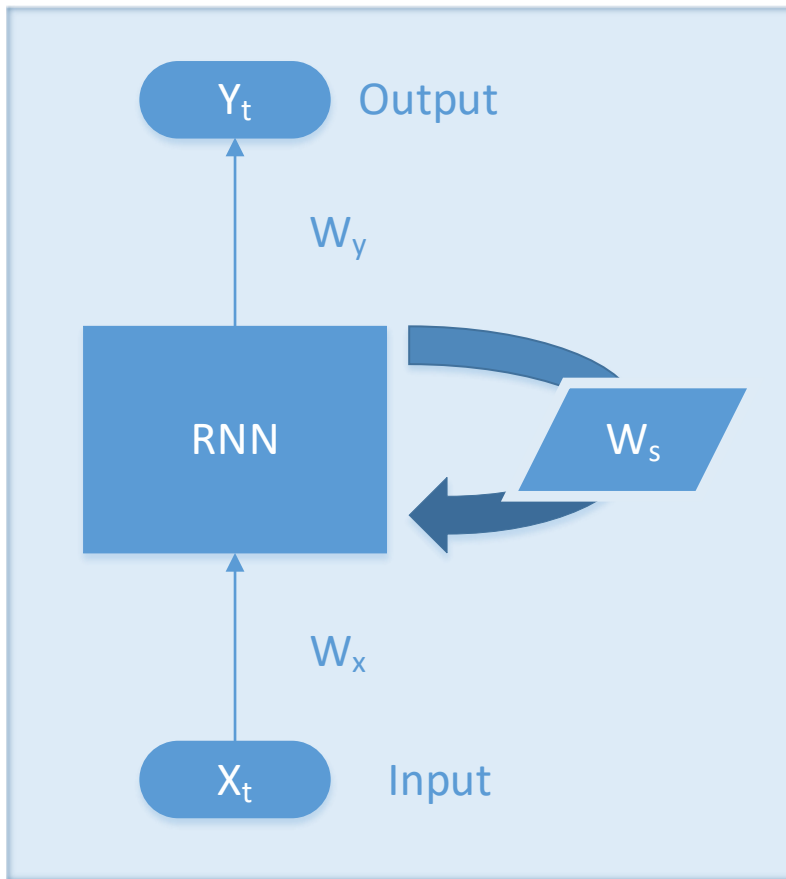


Figure 9: Scheme of the fundamental RNN functionality, own elaboration based on (10)

Figure 9 shows the simplified behaviour of an RNN. The neural network receives an input  $X_t$  and generates an output  $Y_t$ . A loop, here expressed as an arrow, allows to recursively pass information about the weights of the network's layer, represented with a  $W$ . The behaviour of the RNN through the different time states is visualized in more detail on the Figure 10.

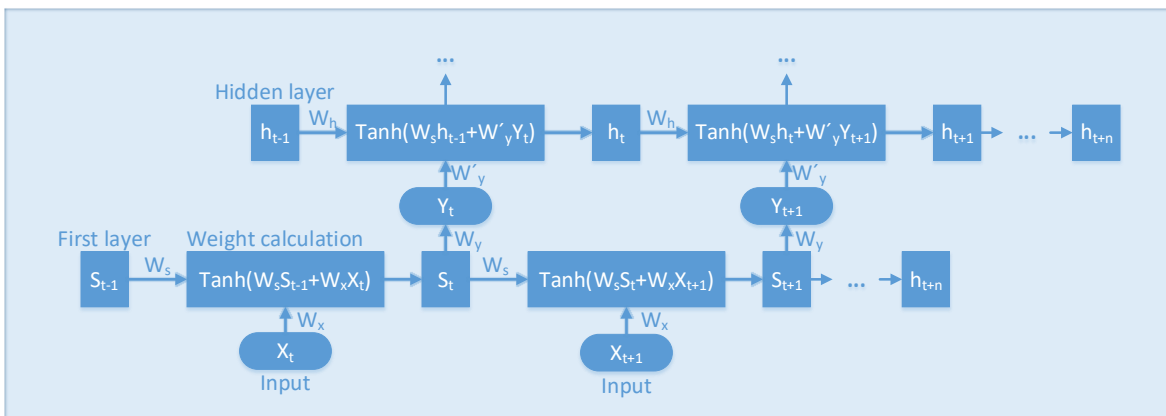


Figure 10: RNN structure in detail, own elaboration based on (10)

The flow diagram Figure 10 demonstrates how the training process works. Shown are two layers, the input layer  $S$  and a hidden layer  $h$ . Both layers change their state at different times  $t-1, t, t+1, \dots, t+n$ . At each time step, the layers will receive an input  $X$  that is used to update the weights of the layer

using, to keep the previous example, a tanh function. Finally, it will generate an output  $Y$  that will represent the input for the following layer.

Figure 11 shows the working structure of an RNN. Each neuron computes the weight for the input  $X$  and if the activation function gives a positive result, it passes on information for the next neuron. Once the input is processed, the network generates an output with a prediction  $Y$ . At that point the model is optimized by applying a loss function that calculates the error of the prediction. Based on this calculation, an optimizer updates the weights to move in the direction of a minimum loss.

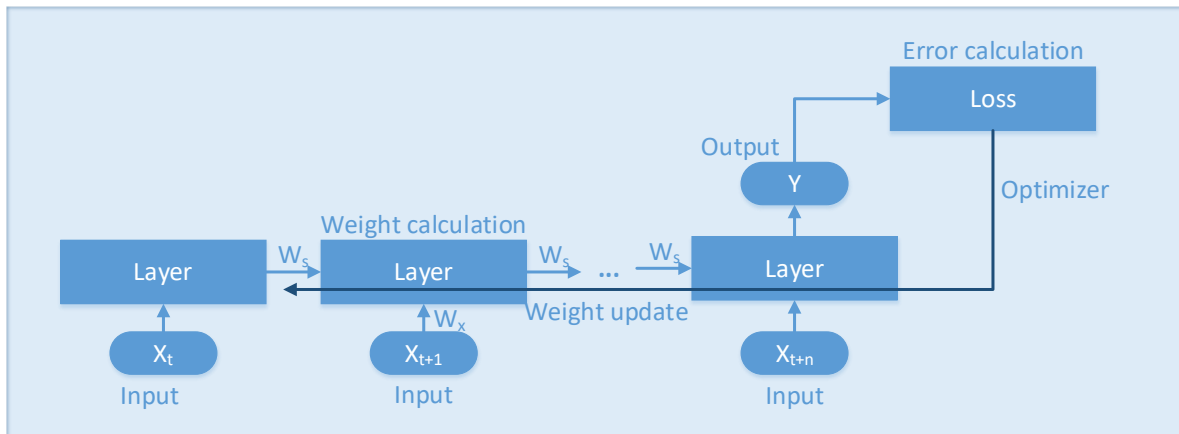


Figure 11: The recursivity in a RNN diagram, own elaboration based on (12)

The recursivity of this network can generate issues because the neurons can be activated to infinite or to zero when iterating several times over the input. This problem happens when the error gradient is too small to train correctly. In the RNN the derivatives of each layer's activation function are looped back from the last layer to the initial layer, unrolling the recurrent calculation of the initial layer values. When selecting the activation function, it is important to avoid multiplying groups of layers with activations that have small derivatives. This is to avoid the reduction of the gradient being dragged to the initial layers, which are determinant to basic recognition of the input. The increase of this gradient can end up in a big inaccuracy on the predictions. (13)

#### 9.2.4 LeNet5

LeNet5 is a basic CNN architecture presented in (14) that uses the back-propagation algorithm also applied for RNN. It is made for pattern recognition, especially hand-written and machine printed numbers and characters. The input of this NN is typically small, conformed by 32x32 size elements. It is considered a good example of gradient based learning technique. (14)

#### 9.2.5 AlexNet

Introduced by (15) and owing its name to one of its creators, Alex Krizhevsky, this is a well-known CNN that won the ImageNet Large Scale Recognition Challenge in 2012. It is pretrained with the ImageNet database and it is possible to directly load the weights of the trained model. This NN works

specially well for training with image data, but it has also been used for training with video datasets, for example in (16).

### 9.2.6 MobileNet

MobileNet is an efficient CNN architecture to build small models that fulfil requirements for mobile and embedded vision applications in (9). To reduce the computation cost, the first few layers of this network have depth wise separable convolution filters, except the first layer that is fully convolutional. As the MobileNet models are small, they have less problems with overfitting, making the regularization and data augmentation techniques obsolete. (17)

### 9.2.7 SqueezeNet

SqueezeNet is a CNN architecture capable to achieve a result similar to AlexNet but having 50 times less parameters (18). This is achieved by replacing the 3x3 filters to 1x1, “squeezing” the channels into 3x3 filters and down sampling late to keep a large activation in the convolution layer. In (9) this network is considered for embedded system use, as it is especially beneficial with limited system resources.

### 9.2.8 VGG16

VGG16 was developed based on AlexNet and uses small 3x3 filters that increase the accuracy analysing characteristics. The advantage of using this model is that it is pretrained with ImageNet and enables to directly load its weights as it can be seen in the code below. (9)

```
from Keras.applications.vgg16 import VGG16
VGG16(weights='imagenet')
```

### 9.2.9 LSTM

The Long Short Term Memory (LSTM) was developed in 1997 and has been used for famous tools such as Google Translate, Siri (Apple) and Alexa (Amazon). Its advantage is to reduce the two problems of the RNNs: the gradient vanishing and the gradient explosion (19). A LSTM neural network is an RNN architecture with the advantage of feedback connections with a structure connection between the cells as shown in Figure 12. It uses LSTM cell blocks instead of neurons which means that its units have a built-in memory cell (16).

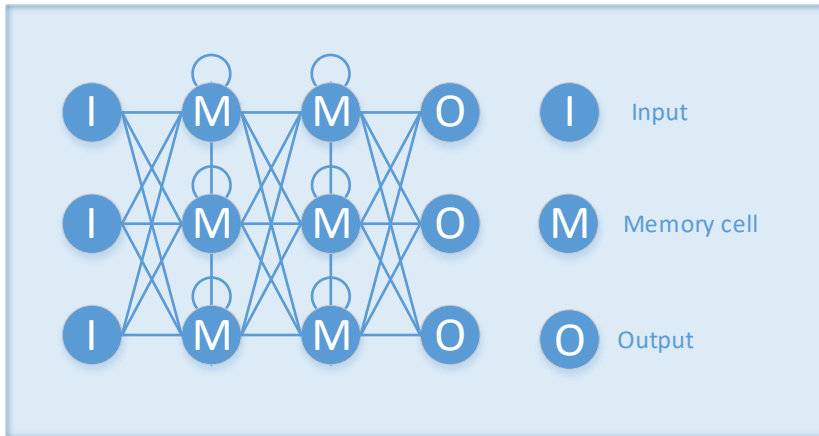


Figure 12: Diagram visualizing simplified the LSTM structure, own elaboration based on (10)

Differing from the RNN, the LSTM architecture uses so-called memory cells (16) more inspired in circuitry than in neuronal connections as can be seen in Figure 13. The objective of the memory cell, a series of non-linear gates used to control how the information flows, is to keep track of the dependencies between the elements of the dataset. They represent simple components called input gate, forget gate and output gate that modify the weight cells. The task of the input gate is to manage the quantity of information that must be remembered from the past in the cell. The forgetting gate, instead, enables to forget non relevant information to release the network from overload problems. As its name shows, the output cell passes on information to the next cell. This structure combines the traditional feed forward NN learning style (long term) with the possibility to connect temporary events (short term) and therefore find common features in dynamic datasets with interdependence. (16)

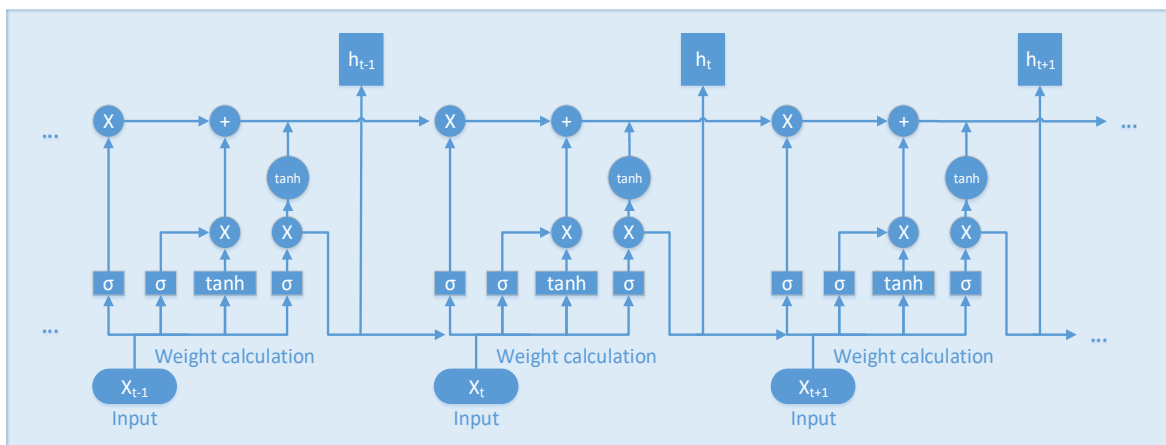


Figure 13: LSTM diagram: own elaboration based on the LSTM mechanism explanations found at (12)

### 9.2.10 Stacked LSTM

The typical LSTM model has one LSTM layer whose output is processed by several different layers. The LSTM layer receives with its input a sequence with previous time steps states. In Keras, it returns the result for the current state not forwarding the time step sequences due to an argument on the



LSTM layer called *return\_sequences*. This argument is a Boolean that conditions whether to return the hidden state or not (Table 11). It is set as false by default.

Table 11: Input and output size possible for a LSTM layer, own elaboration based on the LSTM structure

LSTM	Structure
<b>Input</b>	(batch size, time steps, length)
<b>Output return_sequences = False</b>	(batch size, units)
<b>Output return_sequences = True</b>	(batch size, time steps, units)

A stacked LSTM model contains a group of connected LSTM layers. It makes the model deeper than it would be using only one LSTM layer, each layer processes a part of the task and pass it on to the next layer that will continue working on it making a higher level of abstraction. This way the information about the previous time steps is shared. To stack a group of LSTM layers the argument *return\_sequences* must be activated as true in all the LSTM layers except in the last one.

### 9.2.11 Convolutional LSTM

A Convolutional LSTM (ConvLSTM) layer works similar to a LSTM layer but adding the power of convolutional calculations. It exchanges matrix multiplications through the convolutions and enables filters as seen in Figure 6. Those filters make re-dimensions in the input without losing the original dimension values of the matrix that passes through the layers.

## 9.3 Training Details

This section introduces different settings and approaches to train a NN. Furthermore, is explained how they will affect the learning result.

### 9.3.1 Validation Style

There are different possibilities to evaluate the learning process of a model during the training time. It is determinant to define if the currently learned weights are giving good results with the data. The typical protocol to check the validity of the results implies to separate the dataset in three parts. The training set, as its name implies, is the input used to train the NN. In each epoch it is applied to make the system predict from the input and compare with the expected output. The validation set is used to test how the model is working at the end of the epoch. Finally, the test set is the fraction of data used to evaluate the model’s result of training. Therefore, this set is an independent not being used during training, nor validation. To correctly use this sets, interpreting the learning process through the epochs is important, as the net’s capability of extrapolating the learned information because it may give clues about the learning process and discover problems such as overfitting.

#### 9.3.1.1 No Validation

It is possible to train a model not using a validation dataset. It implies that at the end of each epoch the weights obtained are not used to validate the results with another piece of data.

### 9.3.1.2 Independent Dataset

It consists in loading a validation dataset at the end of each epoch that does not belong to the training set. This option only is interesting if the validation set is a representative selection, otherwise this will cause the problem explained at 10.5.6.6.

### 9.3.1.3 Part of the Training Set

As the name indicates, this method uses a part of the training set to validate the accuracy at each training epoch. It brings the risk of overfitting, explained at 10.5.6.3, as the model is not receiving any new element to generalize over the trained, but only data seen before.

### 9.3.1.4 Cross-validation

This technique consists in reusing the training dataset also for validation. Therefore, the dataset is split in  $k$  parts, at each epoch a different part is used for validating and the other parts for training. This process is repeated as shown in Figure 14 until all parts are used once for validation. The method is also called  $k$ -fold cross-validation.

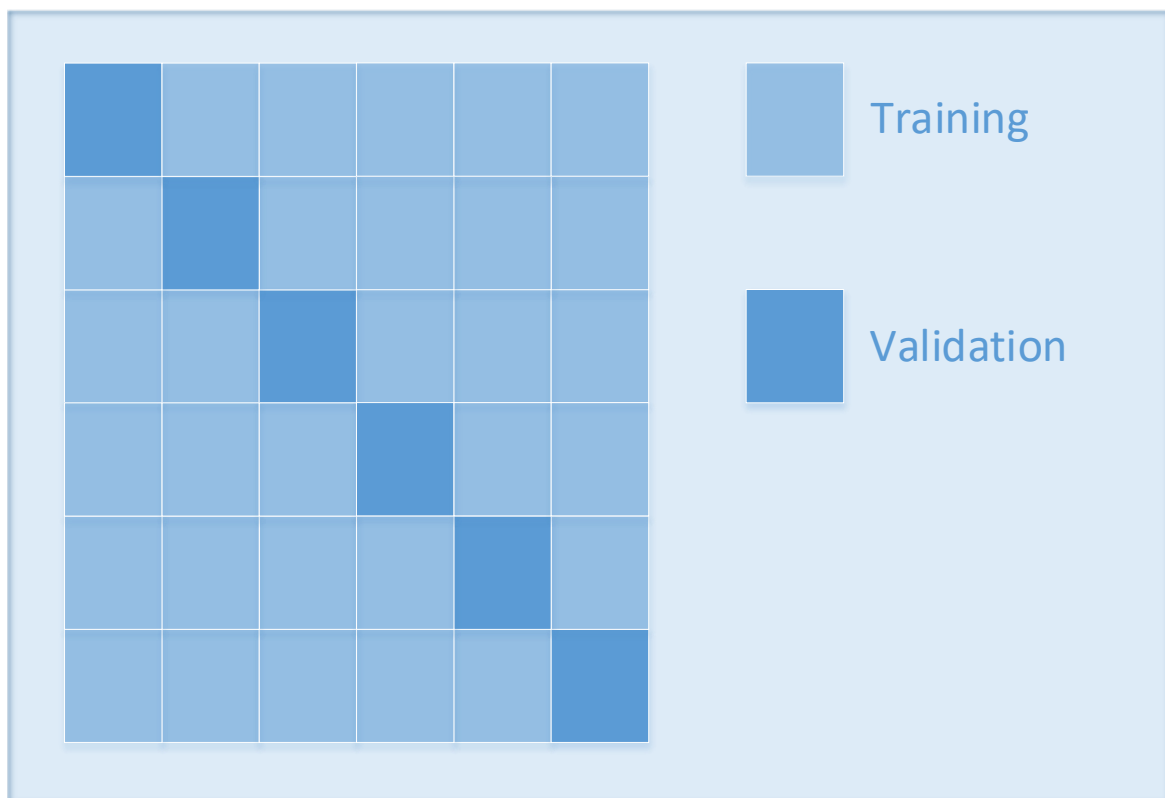


Figure 14: Cross-validation structure, own elaboration based on the cross-validation technique

### 9.3.1.5 One Step Ahead Cross-validation

This technique, explained at the book (20), is used for time dependent models such as LSTM when the training set elements are linear and their time order is crucial. Then cross-validation, which implies removing a section of the training set, is not suitable as the dataset would lack some key information. To apply this method, at each epoch a new sample is added and used to validate the

training with the previous samples as shown in Figure 15. The time order is respected, and a new validation is added on each time step. This is interesting when the training input is not divided in individual elements.

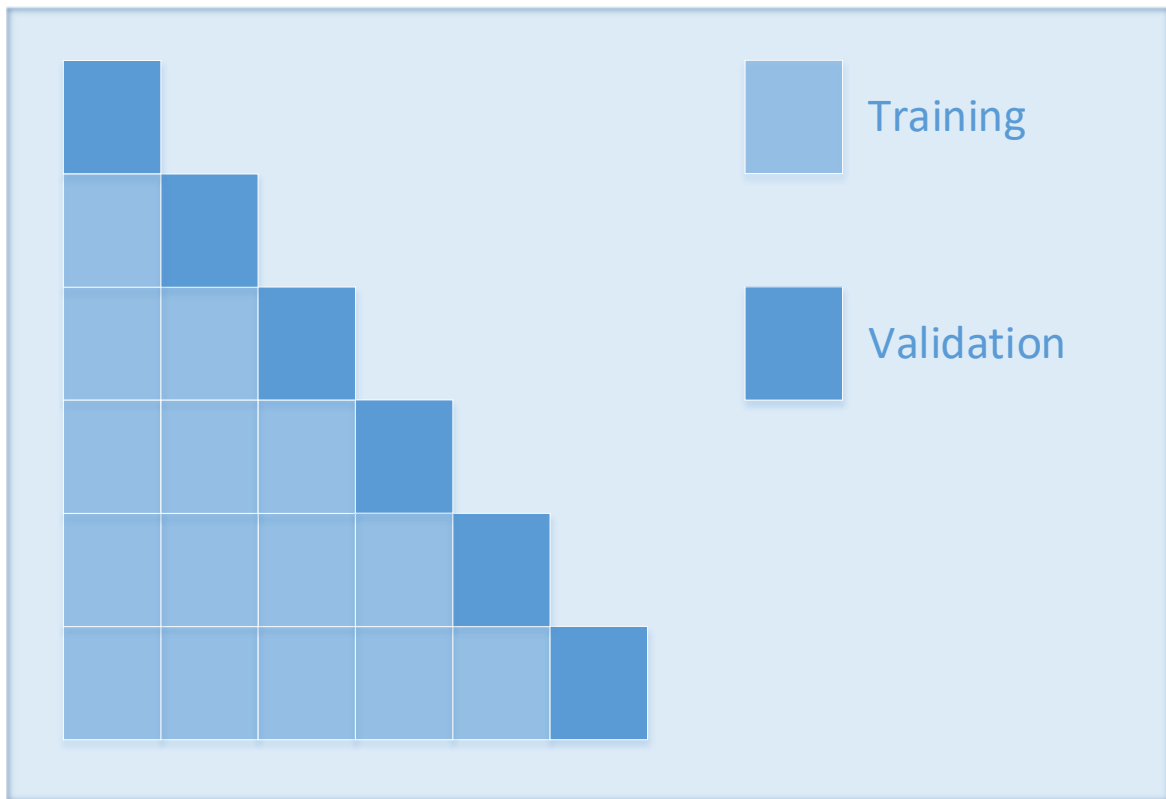


Figure 15: One step ahead cross-validation structure, own elaboration based on the proposal at (20)

### 9.3.2 Optimizers

An optimizer is a method to update weights and learning rate to minimize the loss function. As explained in chapter 9.3.4, the optimizer and the loss are a required input to compile a model from the Keras library (21). Literature introduces various optimizers suitable for specific cases. A selection of them is listed in this chapter.

#### 9.3.2.1 SGD

The Stochastic Gradient Descent optimizer (SGD) replaces the gradient by an estimation, this reduces the computational cost and slows the loss to converge. (22)

#### 9.3.2.2 RMSprop

RMSprop is an adaptative learning rate optimizer that speeds up the mini-batch learning by dividing the learning rate for a weight and making an average of the magnitudes of the recent gradients of that weight. (22)

#### 9.3.2.3 Adam

Adam optimizer is an adaptative optimizer similar to RMSprop based on the estimation of low-order moments, requiring little memory and being recommended for training with large datasets, large

parameters or when the gradients are very noisy (23). Adam keeps track of the average exponential decay of the previous gradients and uses the average of the first and second Momentum shown at Figure 16 for the gradient calculation. (24)

#### 9.3.2.4 Adagrad

Adagrad works well with sparse gradients and converges slower than Adam and SGD (23). It is an adaptative optimizer with parameter-specific learning rates, applied preferably with unmodified parameters (24). The learning rate value decreases its value in relation to how often the parameter is updated during the training. (21)

#### 9.3.2.5 Adadelta

Adadelta is an adaptative optimizer and a variant of Adagrad. It uses a window with the gradient updates to add dimension, this way the window moves through the updates without the need to accumulate the past values. Therefore, Adadelta continues learning after a big number of updates because the learning rate depends on the window values and not on simply decreasing with the updates. (21)

#### 9.3.2.6 Adamax

The Adamax is a variant of the Adam optimizer based on the infinity norm. The difference with that model is that it updates the individual weights scaling their gradients in an inverse proportional scale norm for their individual current and past gradients, this means that the gradients are updated based on the max of the decayed gradients. Therefore, when the scale tends to infinite, the algorithm is more stable. (23)

#### 9.3.2.7 Nesterov versus Momentum

The Momentum method shown in [3] and [4] is accelerating the gradient descent across the iterations accumulating a speed vector that reduces persistently the speed. (25)

$$[3] \quad v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t)$$

$$[4] \quad \theta_{t+1} = \theta_t + v_{t+1}$$

$\mu \in [0,1]$  is the Momentum coefficient,  $v_t$  is the gradient retained from the timestep  $t$ ,  $\nabla f(\theta_t)$  is the gradient at a certain time  $t$ .  $\varepsilon > 0$  represents learning rate,  $v$  represent the speed vector at a time  $t$ .  $v$  at a time  $t+1$  is calculated using the values from the previous timestep as shown in the first equation above. Therefore, the speed vector is constantly changing, getting reduced until it converges.

The Nesterov acceleration gradient, shown at [5] and [6], is like the Momentum technique but follows a different order. It is jumping in direction of the previously accumulated gradient, measures the

gradient and makes a corrective jump. As shown at the following formula, the difference is the update of the vector  $v$ , adding the Momentum coefficient to the gradient. (25) (26)

$$[5] \quad v_{t+1} = \mu v_t - \varepsilon \nabla f(\theta_t + \mu v_t)$$

$$[6] \quad \theta_{t+1} = \theta_t + v_{t+1}$$

The optimization with Momentum creates oscillations in the high curvature vertical direction, the Nesterov method avoids these oscillations and therefore it is more effective than Momentum at decelerating over iterations. Nesterov is then tolerating more values of  $\mu$  (25). In Figure 16 is visualized the Momentum and the Nesterov behaviour.

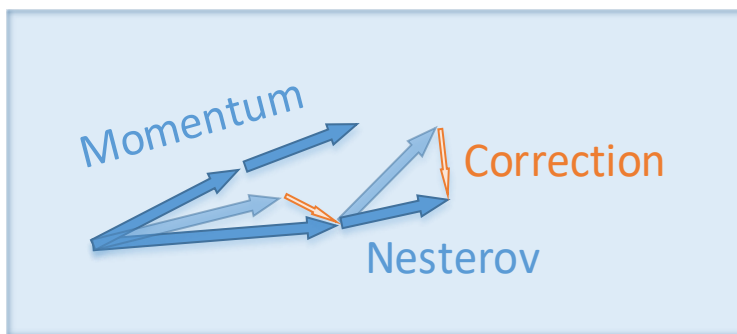


Figure 16: Momentum and Nesterov behaviour, own elaboration based on (26)

### 9.3.2.8 Nadam

The Nesterov Adam optimizer (Nadam) is similar to Adam optimizer, based in RMSprop, but instead of being combined with Momentum, it is combined with Nesterov. Details of the Nesterov method are explained in the previous chapter 9.3.2.7. It is a combination of RMSprop with Nesterov Momentum. (24)

## 9.3.3 Activation Function

In a NN the input of a layer is summed with its weight and then an activation function is applied generating the output that is received by the next layer on the NN. This activation function shall be controlled because it affects the result and accuracy, determining if a neuron is activated and how. In the next sections are the most important activation functions explained.

### 9.3.3.1 Linear Activations

A linear activation function is represented with a straight line in a graph as seen in Figure 17. Assuming a x-y coordinate system, it is calculated by a multiplication applied to x that affects the pendant of the line in the graph, plus a constant that is the value on y when x = 0 as is seen at [7].

$$[7] \quad f(x) = a + bx$$

An identity function is a linear function where the output is equal to the input, applied to the formula above this means a = 0 and b = 1. There is no normalization between a possible range of values and

the value  $x$  is contained between  $-\infty$  and  $\infty$ . Linearity between input and output will complicate the learning process because high values may appear and affect the control feature of the weights.

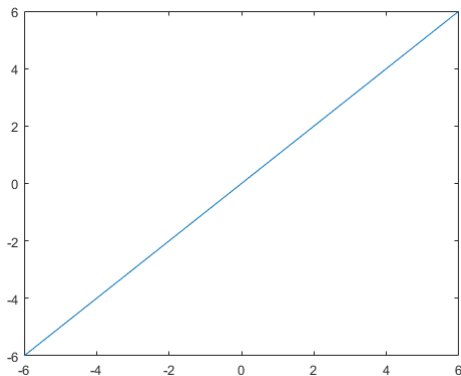


Figure 17: Identity function graphic representation, own elaboration using MATLAB

### 9.3.3.2 Non-linear Activations

Non-linear activation functions permit the NN to compute complex problems using a small range of nodes to represent the output in a neuron. The reason is that these functions, using a x-y coordinate system, will not be represented with a constant line. It is beneficial if it is reduced the pendant in the y axis, simplifying the scale of the result. It also permits to do backpropagation using the derivate.

#### **Binary Step**

This function is useful for binary classification because it makes all positive input equal to 1 and all negative input equal to 0. The mathematic formula is expressed at [8] and visualized in Figure 18.

$$[8] \quad f(x) = \{0 \text{ for } x < 0, 1 \text{ for } x \geq 0\}$$

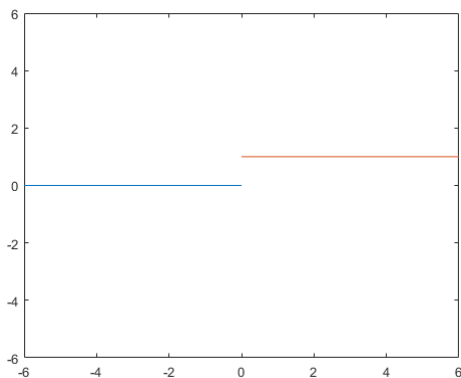


Figure 18: Binary step, own elaboration using MATLAB

#### **Logistic or Sigmoid**

The output of the logistic activation function, with a characteristic sigmoid shape, is a value between zero and one, with big negative values close to zero and positive values close to one. This avoids problems with exponentially growing results ending up on very high values, as possible for example

with the identity function. The function is mathematically expressed in the formula [9] and visualized in Figure 19.

$$[9] \quad f(x) = \frac{1}{1+e^{-x}}$$

For this method is not centred on 0, the model might get stuck during the training if the input has a big negative weight which will force the output close to zero. This can affect the feed-forward activation of the NN, reducing the model parameters updates. This inhibits the model to change its state and create the vanishing gradient problem.

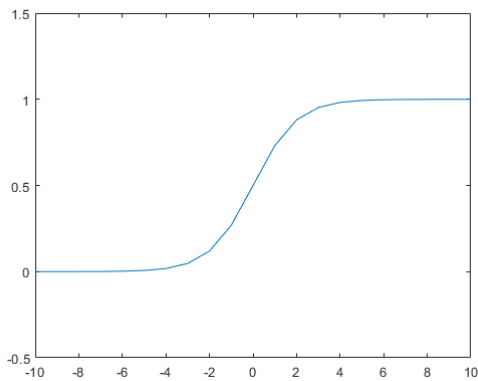


Figure 19: Sigmoid function graphic representation, own elaboration using MATLAB

**tanh**

The Hyperbolic Tangent Activation Function (tanh), formulated at [10], is a sigmoid function expressed with the formula below. It avoids the vanishing gradient problem by having an output that variates in the range [-1,1] as presented in Figure 20. Therefore, even if a big amount of negative inputs shifts the calculation to negative values, neutral values will shift the outputs back to zero, which helps the model parameters to get less stuck.

$$[10] \quad f(x) = \frac{2}{1+e^{-2x}} - 1$$

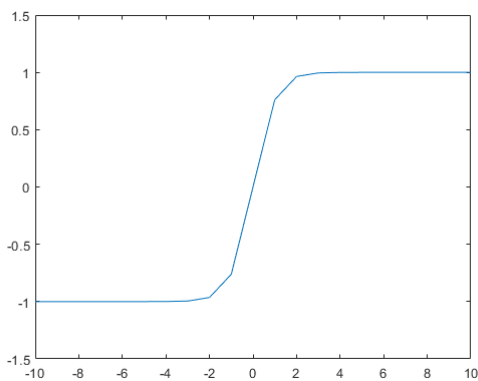


Figure 20: tanh function graphic representation, own elaboration using MATLAB

**arctan**

This is a function that makes a similar distribution to sigmoid and tanh. It gives a value between  $[-\pi/2, \pi/2]$ , incrementing the range of values. Its formula is the inverse of the tangent and shown in [11]. The graphical representation at Figure 21 shows the possible results for the values  $x$  between -10 and 10.

$$[11] \quad f(x) = \tan^{-1}(x)$$

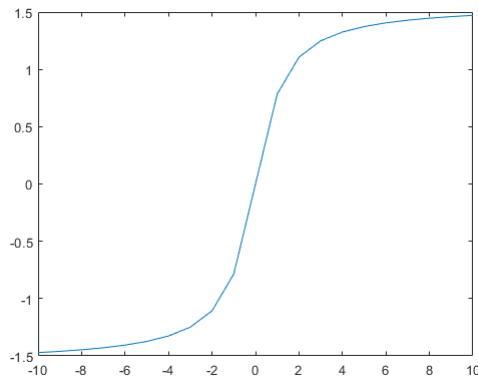


Figure 21: arctan function graphic representation, own elaboration using MATLAB

**Rectified Linear Unit, ReLU**

Rectified Linear Unit (ReLU) is a linear function that outputs the input directly in case of it being positive, otherwise it outputs 0, removing the negative part as shown in the function [12]. The resulting values for  $x$  between -6 and 6 are shown in the Figure 22. Like the logistic sigmoid function, there is the risk that the model stops changing its state if there is too much negative input. It can also stop learning when the learning rate is too high, because it is converging early.

The CNN that are trained with ReLU are proved to train several times faster than those that use the tanh activation function (15). The ReLU does not require an input normalization to avoid saturation. It means as long as some of the model's inputs are positive, the neuron is learning, but local normalization still helps to generalize (15).

$$[12] \quad f(x) = \{0 \text{ for } x < 0, x \text{ for } x \geq 0\}$$



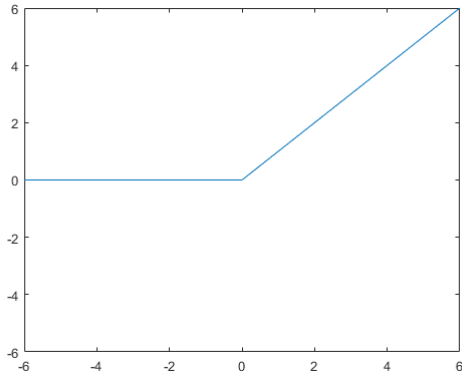


Figure 22: ReLU function graphic representation, own elaboration using MATLAB

**Leaky ReLU**

This is a version of ReLU, but instead of putting the negatives values to zero, it reduces considerably its value and the progression of the negative values decrease slowly. It generalizes better than sigmoid and avoids the neuron from getting stuck at 0.

It has the mean activation close to 0, which accelerates the training (27). As it can be seen in the formula [13] it is mathematically defined as the maximum between x and a small value multiplied per x. Therefore  $\alpha x$ , being  $\alpha$  a predetermined small value, when x is less than 0 and the value when x is higher than 0. The result for values x between -6 and 6 and an  $\alpha$  of 0,01 is shown in the Figure 23.

[13] 
$$f(x) = \{\alpha x \text{ for } x < 0, x \text{ for } x \geq 0\}$$

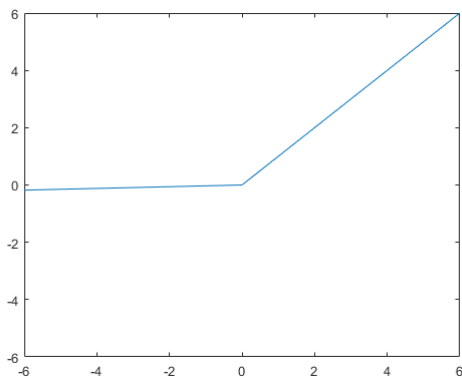


Figure 23: Leaky ReLU function graphic representation, own elaboration using MATLAB

**Parametric ReLU**

Also known as PReLU. Is a variation of Leaky ReLU. Instead of a predetermined value multiplied with x when x is smaller than 0, the value,  $\alpha$ , used to multiply x with is determined by the NN. For this function, shown at [14], a graphic for x between -6 and 6 and  $\alpha=0,05$  is displayed at Figure 23.

[14] 
$$f(x) = \{\alpha x \text{ for } x < 0, x \text{ for } x \geq 0\}$$

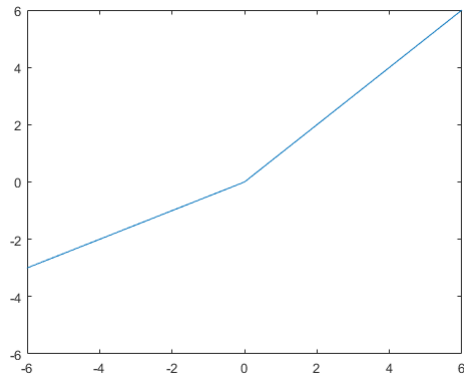


Figure 24: Parametric ReLU function graphic representation, own elaboration using MATLAB

### **Exponential Linear**

Exponential Linear is called (ELU) or (SELU) and is like parametric ReLU, but with a curved shape on the negative side instead of a straight line. It is introduced in (27) and its graphical representation is shown at Figure 25. The function that defines this activation, being  $0 < \alpha$ , is [15]:

$$[15] \quad f(x) = \{\alpha(e^x - 1) \text{ for } x < 0, x \text{ for } x > 0\}$$

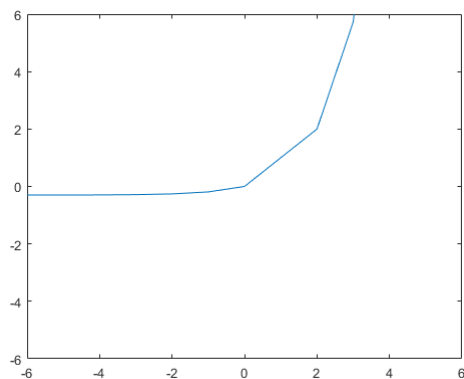


Figure 25: ELU graphic representation, own elaboration using MATLAB

### **Concatenated ReLU**

Concatenated ReLU (CReLU) proposed in (28) is also similar to ReLU, but having two outputs as seen in [16]. One for the positive and the other for the negative input.

$$[16] \quad f(x) = \{[x, 0] \text{ for } x < 0, [0, x] \text{ for } x \geq 0\}$$

### **ReLU-6**

At (29) a study found that for the ReLU activation function, an upper output value limit of 6 gave better results. Therefore, the output of this activation function is limited to 6 for the positive values. At [17] is shown the mathematical formula of ReLU-6 and the graph (Figure 26) that represents the results for a range of  $x$  between -10 and 10.

$$[17] \quad f(x) = \{0 \text{ for } x < 0, x \text{ for } 0 < x < 6, 6 \text{ for } x \geq 6\}$$

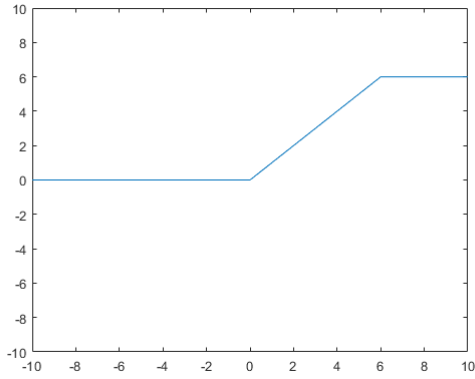


Figure 26: ReLU-6 function graphic representation, own elaboration using MATLAB

**Softmax**

This activation function calculates a probability distribution set in the range between -1 and 1 where the sum of it is 1, from a vector with k numbers, where j represents each element as shown in [18]. This formula is visualized in Figure 27 showing the possible output for an element of the distribution.

$$[18] \quad \sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \text{ for } j = 1, \dots, k$$

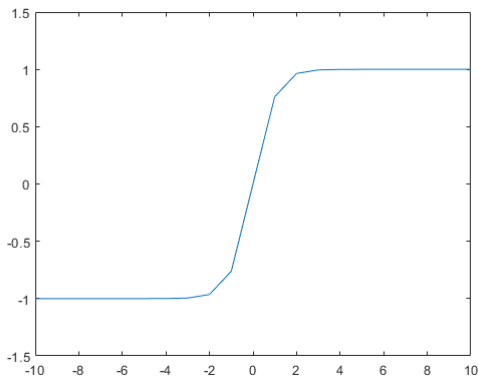


Figure 27: Softmax function graphic representation, own elaboration using MATLAB

**9.3.4 Loss Functions**

The loss function, together with the optimizer described in chapter 9.3.2, is one of the parameters needed to compile a model. The loss is the function used to optimize the score of the model calculating the error. It guides the optimizer to move towards the direction of the minimum. The decision of which loss function to choose should be taken considering the type of data that the model is trained with and the activation function. The loss function can be manually created, but there are existing good defined functions for Keras (30). A selection of important loss functions is explained here.

### 9.3.4.1 Mean Errors

The Mean Error calculations are done regarding the average error committed between the prediction made and the real result. Examples of frequently used functions are defined below.

#### **Mean Squared Error**

Mean Squared Error (MSE) requires that the output line of the model is linear. It measures the quality of the estimation through a mean calculation as shown in [19].

$$[19] \quad MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

The mean of the square of the errors is calculated subtracting the predictions to the real values applying the square to the results and calculating the mean. As seen in the formula above, where  $n$  is the number of elements on the vector that contains the predictions generated,  $Y$  is the vector of values observed on each of the variables that are predicted and  $\hat{Y}$  represents the predicted values. The result of applying this formula is positive, with the best result being 0, because it means there is no difference between the prediction and the actual result. This formula predicts quantities on regression problems. (30)

#### **Mean Absolute Error**

Mean Absolute Error (MAE) is useful when the distribution of the result contains values that are too separated from the tendency. In this case it avoids giving results too small or too big (30). At [20] is shown the formula that defines this loss function using the same nomenclature as in the MSE. It consists in applying the mean to the absolute result of subtracting the expected results to the predictions.

$$[20] \quad MAE = \frac{1}{n} \sum_{i=1}^n |\hat{Y}_i - Y_i|$$

#### **Mean Absolute Percentage Error**

This method is a variant of MAE. It is showing the percentage of the absolute error, which basically is the Mean Absolute Error, multiplied by 100. (30)

#### **Mean Squared Logarithmic Error**

For regression problems where a large value or a group of values needs to be predicted, this method to calculate the loss can be beneficial. It calculates first the logarithm of the prediction before calculating the mean. This way the values are reduced and themselves reduce the difference in error results when predicted values show big differences amongst each other.

### 9.3.4.2 Crossentropy

The MSE loss models have in common the problem of saturation and learning slow. Using a cross-entropy approximation improves the performance of the model, especially those models that have

sigmoid or softmax output. This method uses the Maximum Likelihood Estimation (MLE) framework to find the best estimations of parameters from the experience with previous training data. The crossentropy loss is minimized which means training with small values will bring a better result than training with big ones. (31)

$$[21] \quad \text{Crossentropy} = -\frac{1}{n} \sum_{i=1}^n Y_i \log(x + \hat{Y}_i)$$

This function, seen at [21], represents the average crossentropy of all examples. Where n is the number of elements on the vector that contains the predictions generated, Y is the vector of values observed on each variable being predicted and  $\hat{Y}_i$  represents the predicted values and x is a very small number, for example 1E-15, added to  $\hat{Y}_i$  for avoiding the calculation of log(0). Therefore, the best possible solution is a value close to 0, but reaching a loss of 0 is impossible because of the sum of x. (30)

### **Categorical Crossentropy**

This loss function supports mapping the probability of classifications to match between the different possible labels. It brings a prediction for multi-class classification. The output of this function has the form of a vector that contains the distribution of probabilities of different classes. A good last layer activation function to combine with this loss is softmax. (30)

#### **9.3.4.3 Hinge**

The hinge loss is used for classifying the maximum margin of a classification. Following are three kinds of hinges that might be interesting.

### **Hinge**

Hinge is an alternative to crossentropy that can be used to classify binary problems, where the classification is set in the range [-1, 1]. For this it is important to use an activation that supports this range. Adamax will work well with this loss (30). In the hinge formula, shown at [22], a value between -1 and 1 is given by  $\hat{Y} \cdot f(x)$ , being  $\hat{Y}$  the output given by the model. the result is evaluated as follows: A value < 0 is incorrect, and > 0 is correct, the distance from 0 determines how mistaken or accurate the value is. Hinge returns the maximum between 0 and  $\hat{Y} \cdot f(x)$ , the maximum margin.

$$[22] \quad \text{Hinge} = \max(0, 1 - \hat{Y} \cdot f(x))$$

### **Categorical Hinge**

The categorical hinge is similar to the previous hinge, but adapted for categorical classifications, calculating a vector with the maximum margin of multi-class classifications. (30)

### **Squared Hinge**

The squared hinge loss, as the other versions of squared loss, uses the square of the score in this case to classify the maximum margin. As the normal hinge it works well when it is used with tanh activation. (30)

### 9.3.5 Epochs

The number of epochs defines the total amount of repetitions that the training algorithm processes the entire dataset. The weights of the model are updated at the end of each epoch.

### 9.3.6 Batch Size

The batch size is a hyperparameter used to determine the samples distribution from the used data set before each update in the model. After all samples in the batch were iterated, the NN compares the predictions developed with the samples and compares them with the expected results. Based on the deviation the current error is calculated and used to improve the model. As the NN updates after each batch, the gradient descent of the error is modified so the batch size directly affects the quantity of updates in the error function. Therefore, the batch size affects the optimization of the model. There are three possible types of batch size.

#### *9.3.6.1 Batch Gradient Descent*

First, the Batch Gradient Descent, where the batch size is equal to the size of the training set, it implies that the model is only updated after all the training set is evaluated. Having less updates in the model makes the gradient descent calculation more efficient. It also makes a more stable update that in some cases can result in a good convergence, but in the other cases can lead to a premature convergence.

#### *9.3.6.2 Stochastic Gradient Descent*

In second place, there is the Stochastic Gradient Descent (SGD), which is a batch of size 1. It calculates the error and updates the model for each training example, which has more computational cost than other gradients, descends but improves the error descent. Having improvements very often, which means a faster learning, these updates can create a noisy gradient signal making the model error jump and have an increasing variance over the epochs. Therefore, the noise learning process can help to avoid local minima but also can make it difficult to settle a minimum.

#### *9.3.6.3 Mini-Batch Gradient Descent*

Finally, the Mini-Batch Gradient Descent, where the batch size is equal to a value between 1 and the size of the training set, is the most used batch size. It is intended to combine the characteristics of the previous models to achieve a result that can be robust but also efficient. Choosing smaller values will lead to faster convergence and slightly increased noise, while choosing bigger values result in a slower but more accurate evolution.

### 9.3.7 Time Steps

Time steps contain the value  $k$  used to control how many frames are considered interdependent. Assumed the value  $k$  is 5, for the scene taking place at time  $t$ , the frames  $t_{-1}$ ,  $t_{-2}$ ,  $t_{-3}$ , and  $t_{-4}$  are too considered relevant for the interpretation.

### 9.3.8 Training Host

Training with big datasets might face possible limitations regarding the GPU process capability of common computers. To avoid this, it is important to choose hardware matching the problem size. As explained below, there are two basic hardware approaches for this project. Each accompanied by individual difficulties and opportunities.

#### 9.3.8.1 Computer

A standard computer is an option to be considered when it can afford enough processing capabilities for the training. Hereby it is important to count on a potent GPU, as the computational time cost of using the CPU may be enormous, especially when training with big datasets like a huge number of frames contained in videos. Having multi-GPU training options with a correct division of data between the GPUS is an advantage. Modern GPUs are particularly well-suited to cross-GPU parallelization, as they are able to directly read from and write to one another's memory (15).

#### 9.3.8.2 Servers

Using potent servers of a third party eases the training task, not having to worry about the memory the data requires to be loaded and being able to load big clusters of data simultaneously for processing. Generally, there are two kinds of third-party servers available on the market:

#### **Rented Servers**

It is possible to rent GPU servers. For example, Cherry Servers (32), available with a monthly fee based on the rented GPU. The prices are in a range from 100 euros to 700 euros per month, dependent on the required specifications.

#### **Colaboratory**

As a free option for scientists, researchers, and students it is offered Colaboratory, that belongs to the company Google. This online tool offers an interactive environment than can be shared by different users that consists in a cloud service server based on Jupiter Notebooks where members can add and execute code.

### 9.3.9 Dropout

The dropout was introduced at (33) and with a specific probability sets the output of each hidden neuron to zero. The neurons that are "dropped out" do not contribute to the pass-forward or participate in the backpropagation of the network. Simplified, the dropout supports forgetting a part of what is learned and therefore is especially useful to avoid overlearning (15).

## 10 Methodology

Based on the existing concepts introduced in chapter 9, the following chapter focusses on the selection of suitable tools to achieve the project objectives and the justification of the chosen methods.

In the past years, the demand for automated rating and tagging systems has increased for various applications. Named shall be, for example, YouTube, that gets a huge amount of uploads every day, making it difficult for operators to be able to check the quantity of videos. Not least because of such big companies' interest, action recognition made a lot of progress in the recent past and there is a good base of methods chosen for similar applications. (5) (8) (34) (35)

In comparison, the detection of violent or aggressive behaviour was less studied even if there is a high demand for various applications, as the support of surveillance, rating video online content or parental control (8).

### 10.1 Sensor Input Data

The main possibilities to counter violent behaviour with surveillance are a preventive way via education or deterrence or a reactive way due to evidence and punishment. The chance of intervention while the act of aggression is very low because of the gigantic amount of resources needed for it, especially security personnel. The required human labour force was already reduced significantly by using security cameras to observe risk areas from a centralized security office. Still the headcount of personnel required to permanently supervise all installed security cameras is overwhelming and the costs enormous. For there are not enough personnel, the real time evaluation of the input and detection of all violence scenes is not possible. Of course, the human response rate and time is slow and camera systems lose much of their prevention effect and intervention potential. An automatized system instead would be capable to detect violence situations in real time and drag the attention of the personnel to where it is needed. First, such a system must be able to see and somehow detect the violence situations. For in many locations already installed, surveillance cameras are an obvious choice as suitable input sensor. In this project the chosen feed for the automatized system is video data.

### 10.2 Datasets

A dataset is a compilation of data, in this case video data, which includes examples with or without specific features. A distinction is made between datasets with labelled files regarding the kind of feature they contain and unlabelled datasets, which do not have content-related label. The selection of datasets is determinant for what the system will analyse, because from training datasets the system identifies the features needed to recognise violence in surveillance cameras video input. There are



enough datasets available on the internet that are related with human behaviour. Table 12, below, shows a list of datasets evaluated for this project.

Table 12: Some examples of human action detection datasets and the references of their work, own elaborated list of remarkable datasets

Human Detection Datasets
SBU Kinect Interaction Dataset (36)
INRIA IXMAS (37)
UT-Interaction dataset (38)
Violent-Flows - Crowd Violence/Non-violence Database (39)
UCF-101 dataset (16)
Hockey Fights (8)
The Movies Dataset (40)

Training datasets should be as similar as possible to the real application and regarding the chosen learning style the dataset must be labelled or may be unlabelled. Labelled datasets are preferred for this project to freely decide the learning style. Datasets showing uncomplex scenes can easier lead to positive results because unimportant features are less distracting the focus of the characteristics to learn. Because of this, SBU Kinect Interaction Dataset and The Movies Dataset are discarded for their complexity and size. The UCF-101 dataset has been discharged because it had bad results with LSTM in (16). Four datasets from Table 12 are preselected for this project, Hockey fight database (8) and Violent-Flows database (39) show group violence acts, the first one at ice hockey matches and the second one shows crowded scenes with equal distribution of videos with and without violence. INRIA IXMAS (IXMAS), used at (37), is a dataset focussed on one person. The dataset is big and contains 12 human actions. It shows individuals acting alone from different camera perspectives. It is considered important for this project as surveillance cameras tend to be set in high positions, generally above the heads of the people and not just in a frontal perspective as it happens with other datasets. Table 13 indicates the labels used in the IXMAS dataset and whether they represent violence or non-violence.

Table 13: INRIA IXMAS labels, own elaboration based on the IXMAS label content

Label	Representation	Violence
0	Check camera	0
1	Cross arms	0
2	Scratch head	0
3	Sit down	0
4	Get up	0
5	Turn around	0
6	Walk	0
7	Wave arm	0
8	Punch	1
9	Kick	1
10	Point arm	0
11	Pick up	0

UT-interaction dataset (UT) is introduced in the paper (41) and is also used at (38). The contained videos have a size of 720x480 pixels per frame and show scenes where pairs of people interact with 6 different kind of activities. Three of them are defined violent and the other three not. These interactions are done wearing more than 15 different clothing sets in front of varying backgrounds. The dataset keeps a good distribution between the violence and non-violence classes which makes it suitable for this project. In Table 14 are listed the labels used in the UT dataset and whether they represent violence or not.

Table 14: UT-interaction labels, own elaboration based on the UT label content

Label	Representation	Violence
0	Handshake	0
1	Hug	0
2	Kick	1
3	Point	0
4	Punch	1
5	Push	1

It is important to work with different datasets because a combination is expected to bring improved generalization. The two datasets with the simplest non-mass violence scenes are chosen for this project IXMAS and UT, as they are considered to give clear input for the training. Also, a custom dataset is created to test the model with an independent input. Table 15 shows a summary of the datasets chosen for this project.

Table 15: Characteristics of the chosen datasets, own elaboration

Characteristics	UT	IXMAS	Custom
<b>Used for</b>	Training	Training	Final test
<b>Different camera positions</b>	1	5	1
<b>Different background scenarios</b>	2	1	1
<b>Labels</b>	6	12	10
<b>Violence ratio [%]</b>	50	16,7	50

### 10.3 Violence Detection

The next step is to determine the subject of analysis: The definition of violence. It can be difficult to extract the difference between violence and non-violence, as it may be influenced by subjective impression. In some cases, even humans face problems to correctly separate a real act of aggression from friendly playing. A machine may have the same issue and possibly mix up activities as running or dancing with violent behaviour. Therefore, features have to be found that can be used to securely identify an act of violence. (9) (35)

The difference between the typical datasets for image analysis and the ones used for video is the interdependence between the frames. When working with image datasets, the label is related with a feature of the current image. When processing video, the label of the video might be related with a

feature appearing in only few frames up to maybe all frames of the video. Also, the feature might not be static, detectable in one single frame, but instead may be a change happening interdependent between several frames. Therefore, each frame must be analysed autonomous and related to the previous or following  $x$  frames.



Figure 28: Frames from the IXMAS dataset that belong to a kick labelled video

Figure 28 shows the process and the relation between frames, their interdependency, the importance of time to define violence. With the factor time also comes the order of frames that determines a time-dependent action. In addition, context is needed to clarify if a kick is just a raised leg or an act of aggression.

### 10.3.1 Computer Vision Techniques

It might be difficult for an AI to identify the time-dependent data that is relevant to correctly rate a video; therefore, computer vision techniques can assist focussing on important features. As a first approximation, violent behaviour may be defined as abnormal behaviour. There are existing methods for abnormal behaviour detection (6), but this will not be sufficient because violence is more complex than just a detected abnormality. On the contrary, the majority of examples for abnormal behaviour are non-violent like, for example a person in a crowd wearing short pants in winter.

Focusing on the video, there are more factors that might determine violence, for example the speed of a movement, body contact or blood. (5) applies a technique called OViF (see chapter 9.1.1) to analyse movement speed in relation with violent content. Focussing on the feature speed can also cause problems, for example, rapid movements like running that do not necessarily imply a violent action.

Regarding the feature blood, (42) and (43) are studies that evaluate the presence of blood to detect violence in videos. For some surveillance cameras are recording images in greyscale, difficulties with the integration of this technique and the blood colour detection are expected in this work. Furthermore, blood presence per se is not an evidence of violence, as there are many cases of violence with the absence of blood.

Further studies (compare (44), (45), (46)) focus their investigation on combining audio and video analysis. Audio input is another feature that might help identifying violence but again, certain sounds might not be safe evidence for an act of violence. Common video surveillance cameras also do not register audio and therefore this work is not including audio analysis.

An interesting procedure to approach the video violence recognition is STIP, explained in chapter 9.1.3. At (47) the state of the art is compared regarding different approximations and the conclusion is that techniques as STIP give good results for action recognition. (8) applies this technique to detect violence in sport videos using a hockey dataset. In 2019, in (9) this method is successfully used for violence recognition using DL. In this study the technique is considered optional, implemented only if no good result is achieved using only DL, because the strategy of this work is to approach the problem with the simplest solution.

At (6) spatio-temporal information is used applying the IFV technique (see chapter 9.1.2) to learn about human movement applied to violence detection using cross-validation. The method showed a high accuracy but also very high computational requirements. As with the previous technique, this work tries to achieve a good result with the lowest possible computational requirements, to make the system suitable for the maximum number possible of surveillance machines. Therefore, this method shall not be implemented until necessary.

### 10.3.2 Neural Networks

Summarized, violence is a very broad group of interactions and it is not possible to generally define it with one or few selected features. More promising is the concept of identifying a combination of many features that violence situations have in common. The advantage of NNs is that through the training the network itself learns which features are relevant, taking all available information in consideration: Be it blur on the frame caused by movement, the position of bodies or any other common factor of the training dataset. Therefore, this study focusses on a NN approach to fulfil the project objectives.

#### DNN

In 2003 was proved at (48) that using DNN is more successful than using a large multilayer perceptron to train deeper. The system is consuming this way less neurons but achieving a better result. One of the advantages of using DNN is that these networks by themselves can find features and a solution adapted to the problem (9). Due to the complexity of human behaviour and the multiple factors involucrate in the movement and interaction, this structure is chosen for this project.

The benefit of using deepness at CNN architectures is described at (15) and it is demonstrated that the results are better with deepness included to CNN. Deep CNNs extract features from the images to make predictions and have proved themselves more suitable for big data applications (9). This perspective is implemented for the project to achieve better results in the training and to take more profit of the datasets as they are composed by big groups of images.

### CNN

There are five deep CNNs evaluated for this project, LeNet5, AlexNet, VGG16, MobileNet and SqueezeNet. The CNN architecture **LeNet5** is chosen as a first approximation based on identifying the structure of the human body. Similar as number and letters, the extremities of the human body may be generalized as a distribution of shapes as demonstrated in Figure 29.



*Figure 29: Human shape, own elaboration using as background a frame from the UT dataset*

Based on this assumption, when people are appearing in the frames, there are structures in different positions in the image bending on concrete axis. It is estimated that when, for example, a leg is close to the body of another person it can be interpreted as violence. The analysis of this simple model for violence video recognition is interesting to check if it is capable to deliver good results despite its simplicity and with low computational resource requirements. Therefore, a NN inspired on LeNet5 and adapted for this problem is developed.

Due to its simplicity there is a risk that LeNet5 is not solving this problem, AlexNet, VGG16, MobileNet and SqueezeNet, might serve as further CNN alternative. AlexNet is one of the most important CNN architectures and known for its impact on machine vision. It is possible that this NN solves the issue without the need of a complex time series based architecture, reducing training time cost. Hence, this model is tested in this project. MobileNet and SqueezeNet are considered as an alternative to AlexNet for surveillance machines with a small processing capability. But regarding their similarities, AlexNet is chosen to prove the functionality of this kind of NNs. The VGG16 network, introduced at 9.2.8, can be loaded pretrained with ImageNet, but as its architecture is more complex than AlexNet's. For a simple solution is searched in this work, VGG16 is not used.

### LSTM

At (49) and (50) is shown that RNNs are applied successfully for motion recognition. As the data analysed is structured in video and the human interaction, this approximation is interesting. The

LSTM chosen because its architecture was created to solve problems that can appear using RNNs as explained at 9.2.9. The method is applied in a **stacked** way, as defined in 9.2.10, to bring the benefits of deepness (compare chapter 9.2.1), including the time dimension between the layers, to take more profit of this architecture.

A more complex solution for action recognition in video, which also has a high computational cost, is to apply LSTM networks in combination with 3D CNN, using the descriptors that are generated by the CNN to feed the LSTM network (9). As a proposal to apply this technique and to take most profit of the LSTM, it is decided to create a ConvLSTM network that contains a Convolutional LSTM layer with its output reshaped and fed to a LSTM layer. It is expected to need less LSTM layers when using convolutions, so with this model it is possible to reduce the LSTM layers and changing one for a convolutional LSTM including time dimension on this and other layers.

### **Computer Vision Techniques**

In (51) LSTM is applied successfully for activity recognition and, for it is a similar application, this architecture is expected to deliver good results for violence recognition. Furthermore, it includes the ability to analyse time interdependencies and therefore the integration of computer vision techniques is not necessary when using LSTM or ConvLSTM. For those reasons computer vision techniques were excluded and the focus of this study are the LSTM and ConvLSTM networks.

### **Conclusion**

A range of different NNs is tested in this project. On the first place, focussing on CNN to check if it is possible to avoid the computational efforts of processing the time dimension. A very simple approach here fore is based on LeNet5 architecture while the following step of complexity is to test AlexNet. On the second place, two NNs that track the evolution of the scene, the time steps, are evaluated. The first is a custom stacked LSTM model and after, the convolutional feature of the CNN is included, creating a ConvLSTM model that contains a convolutional LSTM and LSTM layers. The intention is to find a simple model that can solve the problem. The LSTM strategies are expected to be the best options because they have memory cells that enable to remember previous results and take profit of the interdependency of the images.

### **10.3.3 Learning Style**

For the violence detection is not a cluster problem but a classification to detect concrete facts, the UL learning style does not apply and therefore is not considered for the further development. The feedback-based learning style RL could be fed well with the big amount of data available due to the number of surveillance cameras in use, but the system ought to be released to learn from the feedback received by security personnel. This means releasing a system that, at the beginning, makes many mistakes and thereby strongly affects the work conditions of the personnel using it. Having to react to fake alarms and giving feedback, but also missing real violence situations, which would put into

danger possible victims, would heavily affect the trust put into the automatic system and therewith its acceptance on the market.

Counting on a high number of correctly labelled public datasets with human interactions that include violence acts, the style of learning that suits the best the problem is the SL as the model could learn from comparing its predictions with the actual results of a given dataset.

#### 10.3.4 Pretraining

There is a big dependency between the model's performance and the size of the dataset. For this reason, it is interesting to use knowledge transfer in order to avoid overfitting. This means pretraining the network on a large generic database and after retraining the model to make a fine-tuning (9).

In (16) the LSTM is tested ineffective after having good results in the MediaEval. The creators consider the reason is that the LSTM was trained with the UCF-101 which is a dataset very different from the one used before. There is to expect that the results are going to be affected by the dataset used. The team expects improvements on the LSTM model if retrained (16). Using pretrained models is interesting for big deep architectures that otherwise would develop a very high training time effort.

A good base for pretraining is ImageNet, which is an image database organized by nodes that has over 14.000.000 images. Pretraining with the contained images helps the model to identify people, improve the learning rate and therefore make the system more intelligent. Using various datasets and comparing the results is done before in (5) and (35).

Following this idea but creating an alternative, the pretraining of the network in this study is performed with one violence video dataset. After learning with this pretraining dataset, the weights of the model are loaded and reused in the next training step with another independent violence video dataset. The intention is to use a pretrained model to merge the results of multiple training steps for improved violence detection. This procedure is expected helpful to detect violence actions that are staged in various ways and with several backgrounds.

The NNs are pretrained with IXMAS dataset and the weights are used to retrain with UT dataset. This decision follows the idea of pretraining with a larger, simpler and more general dataset. The IXMAS dataset is expected to be a good base.

### 10.4 Tool Selection

This chapter shows the discrimination made to select tools necessary or useful to develop the project modules training comparative studies and alarm system. Most significant choice is the used library and the related programming language. Advantages and disadvantages are evaluated below, and the most suitable option is chosen.

### 10.4.1 Machine Learning Libraries

There are several machine learning libraries that have been examined to be chosen for this project (see Table 16); Torch, Theano, TensorFlow, Scikit-learn and Keras over Theano or over TensorFlow.

Table 16: Deep Learning frameworks of interest for the project and characteristics of them, own elaboration

Library	Application of interest	Complexity
<b>Torch</b>	ML library	Low
<b>Theano</b>	Symbolic compiler for NNs	High
<b>TensorFlow</b>	ML library	Middle
<b>Scikit-learn</b>	Data modelling library	Low
<b>Keras</b>	Library to be run over Theano or TensorFlow/Fast moulding	Low
<b>PyTorch</b>	Library based on Torch/ Debug	Middle

Torch is a ML library based on the language Lua and using CUDA, it is of simple implementation and very efficient. Theano is a symbolic compiler of NNs interesting for investigations and creating optimized NNs. It is good for optimizations of matrix operations and uses the NumPy syntax. TensorFlow is also a ML library that uses CUDA, but it is based on very optimized C++ code. Theano is very complex in comparison to Torch or TensorFlow.

As Scikit-learn is more centred in the modelling of the data than in loading and manipulating the data, it is used together with other tools as a support. As PyTorch is a library based on Torch and Keras runs over TensorFlow, the decision is between Torch and TensorFlow. Both are suitable but due to the bigger community for TensorFlow, the tool is chosen, and Keras is chosen to be run over TensorFlow and simplify its use and have a good control of the variables used for training. In the end, Keras, TensorFlow and Scikit-learn are the chosen libraries for this project.

### 10.4.2 Language

The core of TensorFlow is optimized C++ and CUDA code. But for its use it is offered for C++, Python, JavaScript and Java languages as an application programming interface (API). For Java API is still in experimentation in TensorFlow Core v2.2.0, it is not further considered for this project. Table 17 shows the criteria on which the three languages C++, Python and JavaScript are evaluated.

Table 17: Characteristics of the possible programming languages to use in this project, own elaboration

Characteristic	C++	Python	JavaScript
<b>Level</b>	Middle	High	High
<b>Example of use</b>	Large systems	Data analysis	Web pages
<b>Style</b>	Complex	Simple	Intermediate
<b>Execution</b>	Compiled	Interpreted	Interpreted/Just in time compilation
<b>Advantage</b>	Manual control of memory	NumPy library	Runs in every browser



The advantage of JavaScript is its compatibility with several browsers, making it suitable for multiplatform purposes, even though, the idea of the final interface is not to be run in a browser. C++ language is interesting as it leads to the opportunity to manually control the memory allocation of the elements, a helpful feature if memory is compromised by large datasets. Nonetheless, Python brings more simplicity, as this language is easy to write and read. Furthermore, it gives the opportunity to use the NumPy library, which belongs to this language and allows creating multidimensional arrays and scientific operations on them.

The final decision is to use python for its simplicity and because of the NumPy library. In case of the need to create more optimized code or to run a browser app, more modules in C++ and JavaScript may be added in the future.

## 10.5 Training Methodology

A big part of the time dedicated to this project is employed to study the state of the art regarding DL to determine the decisions taken. For example, understanding and controlling the normalization, the loss function, and the activation to improve the training results is vital for the development of this project. The next segments explain and justify the settings that are chosen to proceed with the NN trainings using Keras.

### 10.5.1 Normalization

To use normalization is to apply a regularization to something; in this project it refers to adjust values that are used in the calculations during the training. There are different ways to apply normalization to the training, such as adding normalization layers to the model or to normalize the dataset.

Batch normalization layers map the input of a NN layer to a smaller size, they are very useful to avoid the exploding and vanishing gradient problem explained at 9.2.3, keeping the values operated within a smaller range, reducing their sparsity distribution and error on jump calculations.

To normalize the dataset used for training its values are, for example, divided by a set number. During implementation, the importance to normalize a dataset to values that are adaptable to the loss function was recognized. So, the output may be easily used for the loss calculation. Generally, it is recommendable to use values not too much separated. In this study the datasets are normalized dividing their values by 256 which is  $2^8$ , because this value could be of use for making the calculations of the NN more efficient for pooling with 2X2 windows.

### 10.5.2 Dataset

The chosen datasets are available for public use, but they are not optimized for the system developed in this work. This section explains the organization and changes that are applied to the dataset to enable their usage with this system and to achieve a better training result.

### 10.5.2.1 Choosing Colour Distribution

The decision for this study is using greyscale instead of colourful images. As mentioned in the methodology at 10.3.1, the blood colour is not determinant and after replacing the RGB colour codification for greyscale, the images are still completely recognizable as demonstrated in Figure 30. Furthermore, the greyscale can help to generalize and bring more attention to shapes and changes of positions than to colours. Using greyscale images provides another advantage: The size of the image is reduced per three. The text below represents an extract of the program showing the image sizes, three times more image files can fit into the same memory space.

Shape of colour image (240, 264, 3)

Shape of greyscale image (240, 264, 1)

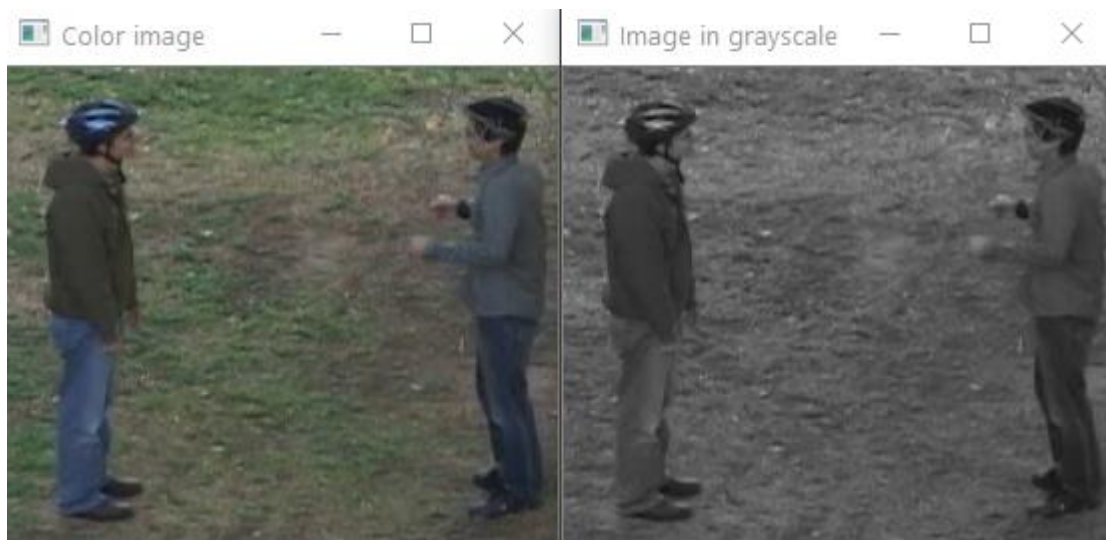


Figure 30: A video frame shown in RGB colour and in greyscale, taken from the UT dataset and edited

### 10.5.2.2 File Changes

To simplify and make the datasets more suitable for the problem, the amount of data is reduced to a representative subset that is enough to give good results but also reduces the computing cost. In addition, the labels are modified to binary violence/non-violence because the goal for this project is to detect violence not to specify the kind of violent action.

### 10.5.2.3 Choosing Validation Styles

Regarding the descriptions made of the possible validation styles 9.3.1, there is the need to choose a suitable one for the project. Table 18 shows a summary of criteria, used to analyse possible styles of validation organization, based on the required style of input and issues predicted when evaluating available methods. The two selected validation styles for this project are, in first place a part of the dataset that is used until the cross-validation is implemented, as this is the method that best suits the problem and avoids the learning problems of the other models.

Table 18: Analysis of the different styles to process the validation dataset, own elaboration

Style	Input	Possible problems
<b>No validation</b>	Elements	Overfitting
<b>Independent dataset</b>	Elements	Validation set unrepresentative
<b>Part of the training set</b>	Elements	Overfitting
<b>Cross-validation</b>	Elements	None seen
<b>One step ahead validation</b>	Constant	The datasets used in this project are not constant

### 10.5.3 Choosing Optimizer

Table 19 summarizes the main characteristics of analysed optimizers that are described at 9.3.2. This helps to choose the most suitable alternatives for this project.

Table 19: List of Keras optimizers with their main characteristics, own elaboration

Optimizer	Main characteristic
<b>SGD</b>	For high redundant datasets
<b>RMSprop</b>	Speeds the mini-batch learning
<b>Adagrad</b>	Converges slow, adapting the learning rate
<b>Adadelta</b>	Learns longer than Adagrad
<b>Adam</b>	For large datasets
<b>Adamax</b>	Better for jump calculations that tend to infinite
<b>Nadam</b>	Like Adam but correcting the jump

SGD is not of interest for this project as, in contrary to the other optimizers, its learning is not adaptative and therefore adapts less to the learning requirements of the model. RMSprop uses mini-batch and as Adam is similar but applying Momentum, it gives an advantage to Adam that also works well with big datasets. For this reason, Adam is chosen as basic optimizer for this system. From the Adam family, Adamax and Nadam are kept in mind as alternatives to Adam, in case problems evolve related with the learning. Adamax is used when the weight values on the scale tend to infinite and Nadam to test the use of Nesterov method.

The learning rate is an important hyperparameter of optimizers. It represents how much a weight is updated within one iteration during the training. It generally is a small value that controls how fast the model learns from the problem, the value is configurable between [0,1]. A small learning rate makes the model learn slowly, on the contrary, a big learning rate increases the speed of the learning of the model. The value of the learning rate should be chosen high enough that the NN learns with sufficient speed, not converging too fast or getting stuck during the learning process. The adaptative optimizers allow adapting the learning rate during the training and therefore help the model to learn correctly. Several learning rates are tested to find an optimum in the range [1E-1,1E-6].

### 10.5.4 Choosing an Activation Function

Based on the descriptions of the activation functions and their characteristics (see chapter 9.3.3), the ones suitable for this project are selected in this chapter. Important characteristics evaluated for selecting the activation function are summarized in Table 20.

Table 20: Summary of the activation function characteristics, own elaboration

Function	Linear	Dying neurons	Vanishing problem	Mean activation close to 0	Learning faster
<b>Linear</b>	Yes	No	No	No	No
<b>Identity</b>	Yes	No	No	No	No
<b>Binary Step</b>	No	Yes	No	No	No
<b>Sigmoid</b>	No	Yes	Yes	No	No
<b>tanh</b>	No	No	Yes	No	No
<b>arctan</b>	No	No	Yes	No	No
<b>ReLU</b>	No/Yes for positive values	Yes	No	No	Yes
<b>Leaky ReLU</b>	Yes	No	No	Yes	Yes
<b>PReLU</b>	Yes	No	No	Yes	Yes
<b>ELU</b>	No	No	No	Yes	Yes
<b>CReLU</b>	No/Yes for positive values	No	No	No	Yes
<b>ReLU-6</b>	[No, Yes],[Yes, No]	Yes	No	No	Yes
<b>Softmax</b>	No	No	Yes	No	No

The linear activation is not used because the linearity is expected to bring problems due to the data complexity. Instead, a non-linear activation is applied, as its output is normalized and helps to generalize. The most common non-linear activation functions are sigmoid, tanh and ReLU. The way they are combined in the model is very important for the performance (52). The binary step is not taken for this system because the human interaction is complex and there is a big number of different actions, this function is expected to be too simple for the problem.

Regarding possible problems with activation functions, the dying neurons effect happens when a neuron always generates 0 because it is stuck on the negative side. A neuron on the negative side makes a plane calculation and has difficulties to have discriminations again. Activation functions with a plane 0 output for the negative side, like sigmoid or ReLU, will more likely have this problem. 9.3.3 shows the functions of the activation functions evaluated. All functions whose negative part is represented as a plain line on the x axis, are more likely to have this problem.

Another problem is the vanishing problem, explained at chapter 9.2.3, it happens with functions that reduce big input on small sizes. This applies to sigmoid, tanh and arctan functions, that reduce the input in a small range between  $[0,1]$ ,  $[-1,1]$ , and  $[-\pi/2, \pi/2]$  respectively. Small changes on the input

imply small changes on the output, and therefore the derivate becomes small (53). A way to reduce the vanishing problem applied in this project is to use a normalization layer explained at 10.5.1.

An advantage of ReLU is that it has less computational cost than sigmoid and tanh, which have expensive exponential calculations (54). Therefore, reduces the training time required. Regarding the size of the datasets used this is very positive.

Having the mean activation close to 0 accelerates the learning: “Mean shifts toward zero speed up learning by bringing the normal gradient closer to the unit natural gradient because of a reduced bias shift effect” (27). It can be useful to use these functions when the one used is converging too early. Leaky ReLU, PReLU and ELU have this characteristic, they can be a good option to improve the results achieved with ReLU. As PReLU and ELU leave to the NN to decide the value of  $\alpha$ , is taken Leaky ReLU to manually control this value. CReLU is not used as it is not contemplated the use of two outputs, and ReLU-6 is not expected to be needed

The ReLU function is the chosen for the hidden layers, if the training converges too soon because of the dying neuron problem, Leaky ReLU or PReLU are the alternative.

Regarding the last layer, is taken softmax as it is suitable for probability calculations and this work aims to give a probability output of violence content. As in 10.5.5 is said that the crossentropy loss combines well with a softmax or sigmoid last layer, sigmoid will be also tested, but expecting a better result with softmax.

### 10.5.5 Choosing a Loss Function

This section provides a summary of facts significant to choose a suitable loss function for the system. Loss functions in general are described at chapter 9.3.4. Table 21 shall give an overview of important details taken into consideration for selecting the loss function used for this project.

Table 21: Summary of loss function characteristics, own elaboration

Loss function	Output structure	Preferable with
<b>Mean</b>	Linear	Sigmoid
<b>Crossentropy</b>	Estimation of parameters	Sigmoid (binary) or softmax
<b>Hinge</b>	Estimation of parameters	Set in the range [-1, 1]

Regarding the output wanted, when only one output per input is wanted, the MSE model is a good, MSE is a mean calculation to use when only one value is expected, this is considered of interest for the binary classification in this project. Instead, for analysing an estimation of different probabilities and then choosing the one that best adapts, categorical crossentropy or hinge would be the best loss functions. Regarding the combination of loss function with activation functions, analysing the state of the art (see chapter 9.3.4.2) the crossentropy is preferred for models with last sigmoid or softmax function layers. The hinge loss, which gives a result within [-1,1], is interesting combined with

activation functions that go on the same range, for example Leaky ReLU. Categorical crossentropy seems to accelerate the learning and as the datasets are complex on their labels and the actions are similar it is considered beneficial that the model gives predictions for each label instead of forcing only one option during training. Categorical crossentropy is selected over the other models for this project in order to accelerate the learning and because the human actions are complex and can have similitudes, therefore, a distribution of probabilities for the possible actions is expected to be more fair to the problem. MSE is also chosen for this project and is compared to check the expected advantage of using crossentropy. The last layer activation function to combine whit categorical crossentropy for a distribution of probabilities is Softmax. To ensure in the NN that the output is a distribution, the labels should be organized using to categorical from Keras.

### 10.5.6 Interpreting the Accuracy and Loss

The learning curves are graphs that visualize the learning performance of a model. They typically visualize the plot of learning on the y axis and the x axis represents the amount of training epochs. These curves are especially useful to rate the performance of ML as the learning rate is incrementing over time (55). In this project the training data is shown using the training history visualization from Keras. The *fit* or *fit\_generator* functions return a history object, from where data collected during the epochs is taken. The accuracy and the loss evolution through the epochs are valuable data for evaluating the model's performance and detect learning problems. One simple way to visualize the results is using matplotlib library to generate graphs with the evolution results. Table 22 shows the different elements to consider when interpreting an accuracy and loss graph.

Table 22: Tools to interpret the results of the learning on a NN, own elaboration

Tool	Description
<b>Accuracy</b>	Performance learning curve, this metric shows how correct the model works
<b>Loss</b>	Optimization learning curve, how the model is optimized
<b>Train learning curve</b>	Calculated with the training dataset. To see how the model is learning during the training process
<b>Validation learning curve</b>	Calculated with the validation dataset, to see how well the model is generalizing with data not trained with

In this project two graphs are generated after each training. The first graph shows the accuracy during training and validation and the second graph that shows the loss during training and validation. It is important to understand the typical behaviours of a learning curve to diagnose how the model is learning and to enable fast troubleshooting.

#### 10.5.6.1 Not Learning

A model is not learning when its accuracy does not improve and the loss is not reducing over the epochs, which means that the errors are staying constant, except maybe some noise. Figure 31 and Figure 32 present typical graphs of a non-learning model.

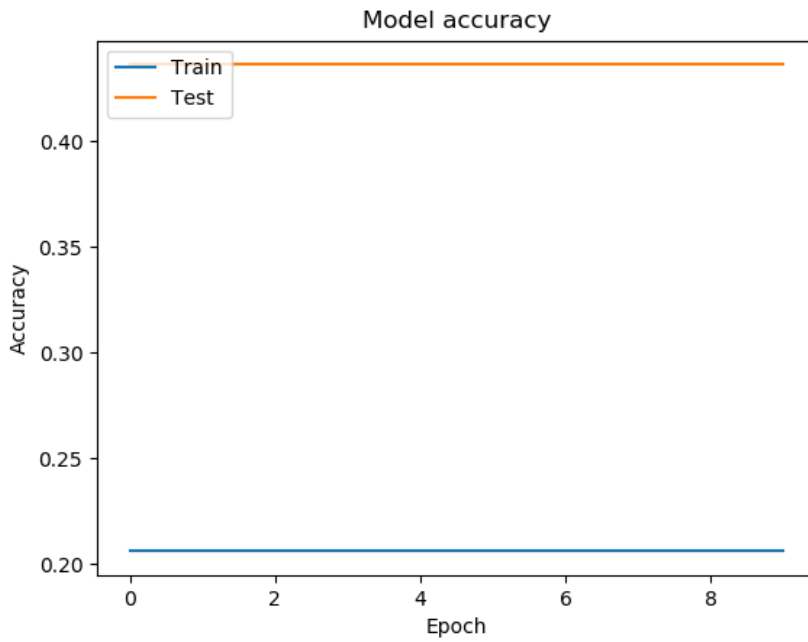


Figure 31: Accuracy evolution of a non-learning model, created using matplotlib

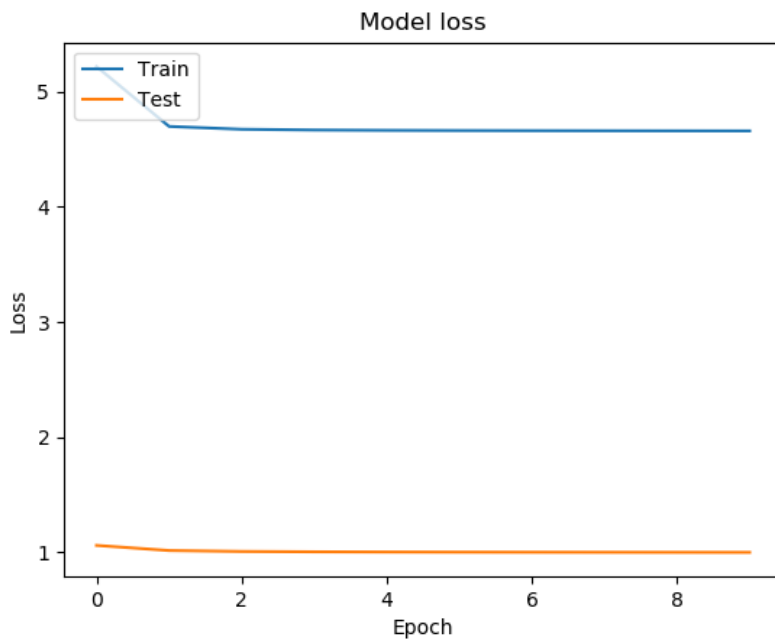


Figure 32: Loss evolution of a non-learning model, created using matplotlib

### 10.5.6.2 Underfitting

The underfitting occurs when a model is not capable to learn from the training dataset. This can be identified when analysing the training loss graph as a flat line or noisy values of high loss. Also possible is a training loss that continuously decreases during the training and shows a tendency to continue decreasing at the end of the training. This shows that the model is capable to learn more, and it should learn longer time.

### *10.5.6.3 Overfitting*

Overfitting occurs when a model learns too much from a dataset during the training, including random noises, this means that the model has memorized not determinant features and therefore the generalization capability gets reduced. To observe this behaviour the validation accuracy should be analysed. It happens if the model has higher learning capacity than the one required or has trained too many iterations. It can be identified in the loss graph, as the training loss continues decreasing but the validation one stops decreasing at certain point and starts increasing.

### *10.5.6.4 Good Fitting*

This is the best outcome possible and it is set in between the under and over fitting, having a model that is capable to learn and to generalize. It can be recognized when the graphs of the training and validation loss decrease until they reach a stability with a minimum distance between the train and validation values.

### *10.5.6.5 Training Set Unrepresentative*

When the training set is not wide enough compared with the validation dataset, it happens that the model is not receiving sufficient information to learn about the problem, even though it fulfils the requirements for a good generalization capability. This case can be recognized when both lines of the loss are showing progress but the distance between training and validation is big.

### *10.5.6.6 Validation Set Unrepresentative*

This problem happens if the validation dataset is not giving enough information to the model to learn how to generalize. It can be identified when the learning rate seems to be correct, but the validation loss has strong noise movements along the training loss. Another possible result can be the gap between both curves, training and validation loss, being too big. With better results for the training it can mean that the validation dataset is not representative. Examples for both possible results are demonstrated in Figure 33 and Figure 34.



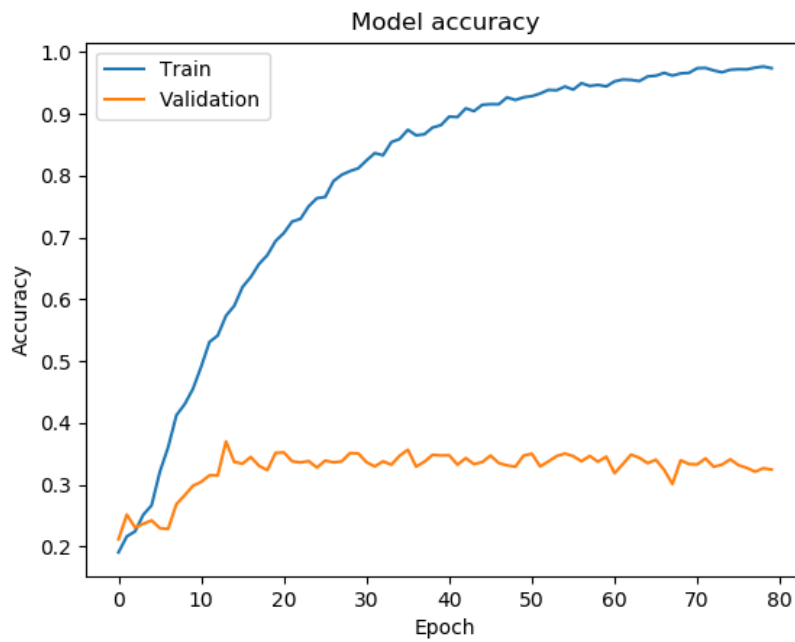


Figure 33: Accuracy of a validation set unrepresentative during the training of a ConvLSTM NN, created using matplotlib during the tests made in this project

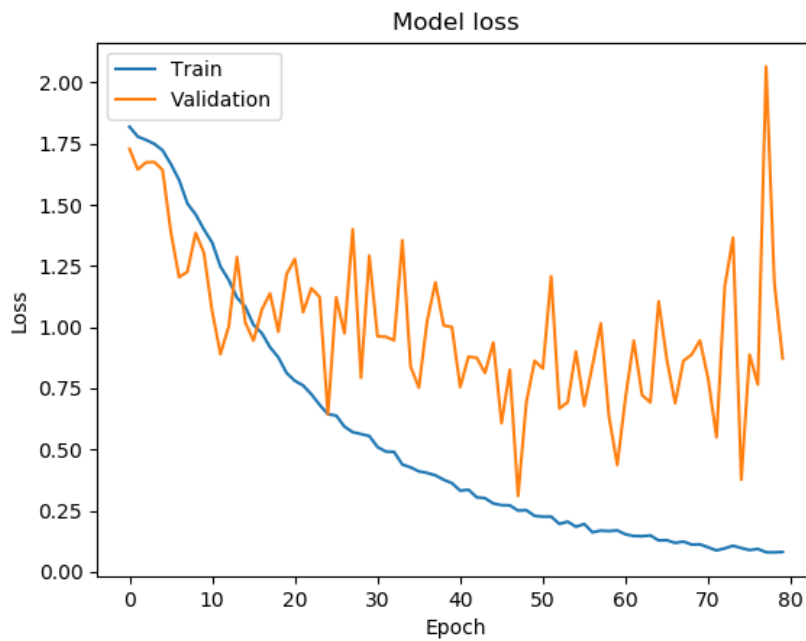


Figure 34: Loss of a validation set unrepresentative during the training of a ConvLSTM NN, created using matplotlib during the tests made in this project

### 10.5.7 Choosing Number of Epochs, Batch Size and Time Steps

The epochs are a key element for training described at 9.3.5. The quantity of epochs used should be enough that the accuracy reaches its top and the loss is stabilized, but avoiding extra trainings that would lead to overfitting as explained in 10.5.6.3.

For big datasets it can be difficult to apply the batch gradient descent because of the spatial complexity of keeping all files of the dataset in memory. Over this method and using a batch of 1, The mini-batch style is expected to be a better approximation because it uses a size of batch and it is possible to adapt its value to control how the model converges bringing robustness and efficiency as said in 9.3.6.3. This adaptability is beneficial because it determines after how many consecutives predictions is calculated the error committed. Therefore, this method is applied and is used with Adam, Adamax or Nadam optimizers that were selected in 10.5.3.

In (56) recommend batch sizes power of two, such as 32, 64, 128. It could help fitting the requirements of the processor used for loading and training the dataset. This setting is accelerating the processing time of a training. The value will be adjusted between 20 and 80 in the tests to find the one that best suits the problem, and a power of 2 in this range will be also compared to determine if it brings improvements for the training.

The amount of memorized time steps required to detect the time-dependent action is relies on the application and therefore there is no fixed rule. (57) recommends to memorize 5 or 10 time steps. When the value gets higher the representation taken to be memorized are a higher subset of the dataset, which also may cause problems because a high value  $k$  means remembering more information that may be not related to the action. When the memorization is applied over a smaller number of frames, the effect of the technique has less effect on the learning, which means that the trained AI less considers the interdependency. For the following experiments, the time step values  $k$  of 2, 10 and 20 are used to find the best option and confirm if the values between 5 and 10 indeed bring the best result.

### 10.5.8 Choosing a Training Host

Several options to host the training are shown at chapter 9.3.8. The possibility of rented servers is discarded due to the costs. To ensure high data security, it is decided to guarantee local training and prepare single and multi GPU mode instead of using Colaboratory servers. This decision increases the training time but brings the experience to confront a limited system and to learn how to take the most profit of it.

### 10.5.9 Computer Characteristics

The characteristics of the hardware used during training is shown in Table 23.

*Table 23: Characteristics of the computer used during the training, own elaboration*

Characteristics
RAM 16.0GB
Processor Intel(R) Core(TM) i7-4790k CPU 4.00GHz
GPU GeForce GTX 980

### 10.5.10 Spatial and Computing Adaptations

Some preparations are necessary to correctly train the NNs in this project. On the first place, it is necessary that the system uses the GPU power, in second place, the project shall enable to use multi-GPU power to take profit of further computer improvements. Finally, the kind of generator used to load the data in memory batches is chosen for the different input dataset structure required by the NNs.

#### 10.5.10.1 GPU Training over CPU

Using GPU over CPU for training big data is very important. Due to the different architecture of both processors, a CPU can quickly fetch small amounts of packages in the RAM, the GPU is specialised in fetching big amounts of memory data in a slower way. Due to the quantity of videos to be analysed there is a great quantity of frames to process in this project. This is just what the GPU is specialised for and using it instead of the CPU is significantly improving the processing speed.

#### 10.5.10.2 Preparing the Computer for GPU Training

In this section is explained how to prepare the software to use the GPU for training. The package TensorFlow does not permit to train multi-GPU, instead the tensorflow-gpu should be installed in the project interpreter because this package allows to use TensorFlow over the GPU.

The next step is to install CUDA as a platform to compute the GPU on the project. During the testing of the tool a problem was found: TensorFlow could not use CUDA and the function *cudaGetDevice* failed. The result of the root-cause-analysis is that the used version CUDA 10.1 does not include the *cuda64\_100.dll* that is required by TensorFlow 2.0 (58). As work-around it is possible to either manually add the required dll-file to the bin path in the installation folder or to install the supported version CUDA 10.0 (59). To avoid further incompatibilities, CUDA 10.0 is used in this project.

To integrate the CUDA Toolkit Visual Studio is installed. It is important to install a version of Visual Studio that is supported by CUDA. For this tool combination Visual Studio Community 2017 is supported and offers an interesting free version of the development environment.

The next requirement is cuDNN the CUDA NN library. It is necessary to sign in on the Nvidia developer webpage to download this GPU-accelerated library of primitives for DNNs and include it in the CUDA files.

#### 10.5.10.3 Preparing the Code for Multi-GPU Training

To adapt the system to use the multi-GPU power and make a correct multiprocessing Keras offers an option to integrate this function as shown in the code below.

```
print(_get_available_devices())
    num_gpus = tf.contrib.eager.num_gpus()
    print("num gpus: ", num_gpus)
```

```
model = tfk.utils.multi_gpu_model(model, gpus=num_gpus)
```

#### 10.5.10.4 Generator Style

To enable local training with big datasets, it is required to load the elements of the dataset by batches of temporal data, keeping the data in memory only while being used. A data generator solves this issue and provides the batches. The first example is the ImageDataGenerator, it is a Keras generator class used to create data batches shaped for the typical training input for models that train with images. It gives the opportunity to add data augmentation to the dataset. A second option is the TimeSeriesGenerator. This Keras class transforms the input data, such as video images, into the shape required for the training of time factor dependent models, as LSTM. The problem with both options is that one generates batches with a structure good to feed images to NN and the second to feed time dependent elements. In this project both structures, and in addition the ConvLSTM structure input, are required.

The favoured option for this work is to develop a custom data generator that creates data batches with different shapes, bringing a reusability value for training different NNs. This custom generator can cover all cases, gives the possibility to customize and adapt and can be of use for future work. At Table 24 the important factors considered when choosing the generator style are summarized.

Table 24: Options proposed to solve the lack of memory to load the dataset while training problem, own elaboration

Options	Data shape	Reusability	Implementation time cost
<b>ImageDataGenerator</b>	For image data models	Less	Low
<b>TimeSeriesGenerator</b>	For time dependent models	Less	Low
<b>Own generator</b>	Can create different ones	More	High

## 10.6 Alarm System

The tools used to develop the Alarm system are the ones determined at 10.4, using python language for its simplicity, and TensorFlow and Keras to load the trained model in order to analyse the frames.

This part of the project will consist in a system that can process video input in real time and detect violence scenes using the developed model. When violence is detected, relevant frames are saved as evidence while an alarm is generated to inform the authorities and enable a quick reaction of the personnel. For this the system must be simple and fast, which means cosmetic features of the software are not necessary. This is a professional tool, clicking buttons or appearing pop-ups on the screen can interfere the view of or unnecessarily distract the personnel and therefore, are not wanted. After detailed analysis, it was concluded that subtask “Frame the people on the scenes” is not implemented in the system. Automatic generated frames do not add extra value but could distract personnel or in the worst case even disturb the view on significant happenings.

Combined with the sound alarm when detecting violence, the scene shall be visible from a video frame where the camera ID is written for a fast localization of the event. The personnel shall not need to interact with the program and instead focus on interfering in the act. To avoid unnecessarily occupied storage space, the evidence frames are saved in greyscale, not in colour. The name of the camera used, date and hour are added to the saved frames to add reliability and avoid mistakes evaluating the evidence.

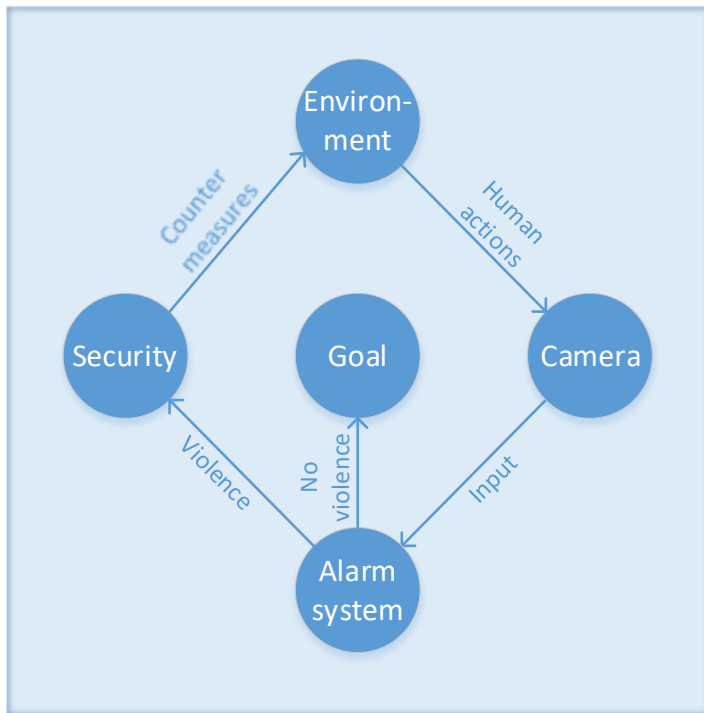


Figure 35: Behaviour of the alarm system, own elaboration

### 10.7 Summary

This project studies different approaches for violence detection in videos based on NNs, more specifically, a LeNet5 model and AlexNet. If those NNs are not sufficient to solve the problem due to the time dependency between frames, the use of LSTM and convolutional LSTM layers is realized. Two datasets are used and, following the advice from (16), the possibility of improvement using pretraining is applied. The training structure is analysed and set, and the trainers created. A custom data generator needs to be created to provide data batches with different shapes suitable for all applied NN architectures. Finally, a simple real time violence in video analyser software is developed and tested with an independent dataset.

## 11 System Description

This section describes the characteristics of the implementation. First, the used software is named and, the two modules implemented to fulfil the objectives of this project are described: The training comparative studies and the alarm system. The design of this system using unified modelling language (UML) used to create this system can be seen at the annex 16.3.

## 11.1 Used Software

The system is built using Python language over PyCharm environment developed by JetBrains. The main external libraries employed are TensorFlow, Keras, OpenCV, matplotlib, NumPy and Scikit-learn. To enable the use of the GPU for the training calculations, CUDA and Visual Studio Community 2017 have been installed. For the project documentation Microsoft Office is applied, in particular: Word, Excel, and Visio. Graphs are created with MATLAB.

## 11.2 Training Comparative Studies

The module “training comparative studies” is created with a **scalable** perspective, to make possible to include models with three different structures of input data. The trainer class and custom generator can use different networks and structures of data and it is simple to add new ones. This makes this code also **reusable** for different projects and purposes. Heritage of classes, enumeration of model types and a custom generator that can adapt the batches to the required structure are key for the successful development.

### 11.2.1 Models

The NNs code is separated on different python files depending on the model type and called by the trainers to load the model. Here are described the NN developed in this project. To generalize, all models receive the input and output size by parameter to make it adaptable to different problems. Also, it is possible to load an already trained model instead of creating a new one. Below are described the general structure of NN systems tested. The models UML can be seen at the annex Figure 85.

#### 11.2.1.1 CNNs

Both used CNN models are included on independent files. They are based either on the **LeNet5** or the **AlexNet** structure. The shape of the layers of the LeNet5 is adapted to bigger images than the ones typically fed to the net to preserve details of the video. In both cases the activation used on the layers is ReLU except on the last layer where softmax is applied as explained in chapter 10.5.4. The input distribution for these NNs follows:

(samples, channels, rows, cols)

The inner architecture of AlexNet is not explained in this section, as it was not modified for this project. To adapt the net to the complexity of the problem, LeNet5 was customized: Three layers instead of two convolutional 2D layers are used in this NN. The quantity of filters per convolution is also changed from 6 and 16 filters to 16, 64 and 64. The first kernel is of size (7,7), the rest is the same as in the original LeNet5. Furthermore, there are two instead of three final dense layers, but with more elements. The first dense has 1024 units and the second has a variable number of units and this way can adapt to the output size required by the problem. In addition, max pooling is used

instead of average pooling, as it extracts the most important features instead of smoothing the data. The code below is used to create the LeNet5 based model:

```

model = Sequential()
model.add(Conv2D(16, (7, 7), strides=1, padding='same', input_shape=dim, activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), strides=1, padding='same', activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), padding='same', activation='relu'))
model.add(Flatten())
model.add(Dense(1024, activation='relu'))
model.add(Dense(outputshape, activation='softmax'))
    
```

### 11.2.1.2 Stacked LSTM

This model uses a recurrent approach applying stacked LSTM. It makes the model deeper than only using one layer because each layer processes a part of the task before it is passed on to the next layer that will continue working on it and creating a higher level of abstraction. The most suitable stacked LSTM model created in this study has four LSTM and two dense layers. Figure 36 visualizes its basic architecture.

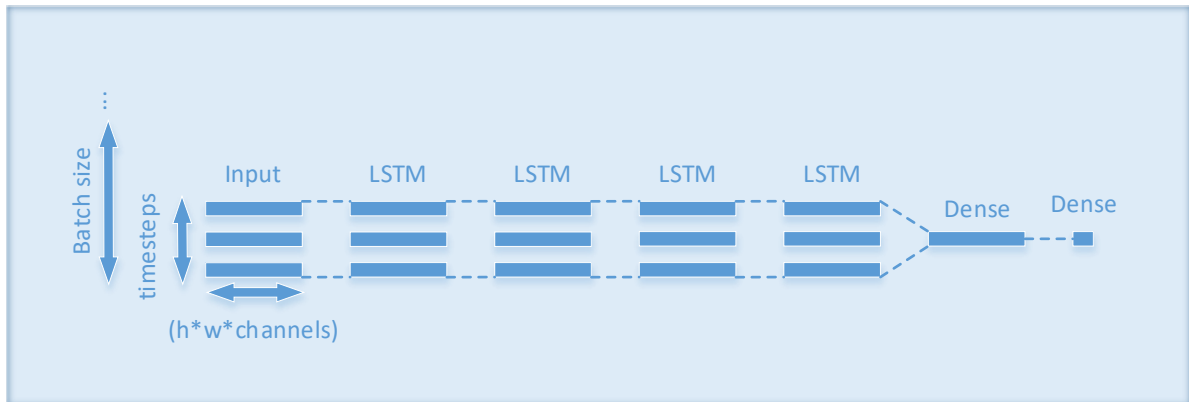


Figure 36: The stacked LSTM model network structure created, own elaboration

This model receives different parameters to generalize its use. For example, a Boolean determines if the weights of an already trained LSTM model should be loaded. For this, the model must have a h5py format. If no trained model is loaded, a new one is created using several variables to make the model adaptable to different dataset sizes and required outputs. The model is sequential, this means that a list of layers is passed to the model constructor. In the code the layers are added using the order *method.add*. Regarding the LSTM layers, As the first parameter of the LSTM layers it is written the number of units named length, that represents the number of elements in the training. Written as first parameter of the LSTM layers is the number of units named length, which represents the number of elements in the training. *Sequential* requires that in the first layer input shape is passed as an

argument. It can be a tuple, or, in case accepting any shape is wanted, it should be defined as “None”. For LSTM layers the input must be given in three dimensions that must be organized, as shown in the code below, with the number of samples, the time steps to be memorized and the size of an element from the dataset.

(samples, time steps, features)

For the next layers the input shape is inferred, so it is not required to specify it. The LSTM output by default is two-dimensional, because it learns from previous time steps but gives a result for the current one. When stacking LSTM, the next LSTM layer will again expect a three-dimensional input, therefore *return\_sequences* must be set to true in all LSTM layers but the last one, where it is set to *false*. Setting this variable to *true* makes the hidden state output visible for the next layer, so the input in the next time step is the full sequence returned by the previous time step. The last LSTM layer only needs to return the result for the current time step. The activation by default is tanh, and the recurrent activation is hard sigmoid. The last LSTM layer output is used in a dense layer, that organizes the input that receives to a tuple, in this case the output is a tuple with shape batch size. Finally, a second dense with the size of the number of classes is applied. The explained settings are applied in the code below.

```
model = Sequential()
model.add(LSTM(length, input_shape=(timesteps, patch_window[0]*patch_window[1]),
return_sequences = True, activation = relu))
model.add(LSTM(length, input_shape=(timesteps, patch_window[0]*patch_window[1]),
return_sequences = True, activation = relu))
model.add(LSTM(length, input_shape=(timesteps, patch_window[0]*patch_window[1]),
return_sequences = False, activation = relu))
model.add(Dense(length, activation = 'relu'))
model.add(Dense(outputshape, activation = 'softmax'))
```

To illustrate the distribution of data passing in the NN, Table 25 shows the size distribution of data for a batch size of 32, 10 time steps and 7 classes. The first three LSTM layers share information regarding the time steps to the following layers while the last one does not share it. The first of the following dense layers has the size of the batch and last one provides the final output, which is the quantity of possible classes.



Table 25: Layers size output of the LSTM NN, own elaboration

Layer type	Output shape
lstm_1 (LSTM)	(None, 10, 32)
lstm_2 (LSTM)	(None, 10, 32)
lstm_3 (LSTM)	(None, 10, 32)
lstm_4 (LSTM)	(None, 32)
dense_1 (Dense)	(None, 32)
dense_2 (Dense)	(None, 7)

### 11.2.1.3 ConvLSTM

The convolutional LSTM-NN (shown in Figure 37) processes the input first in a ConvLSTM layer and after reshapes the data for the use in a LSTM layer. After two dense layers a tuple is returned containing the shape defined by the quantity of labels for the problem.

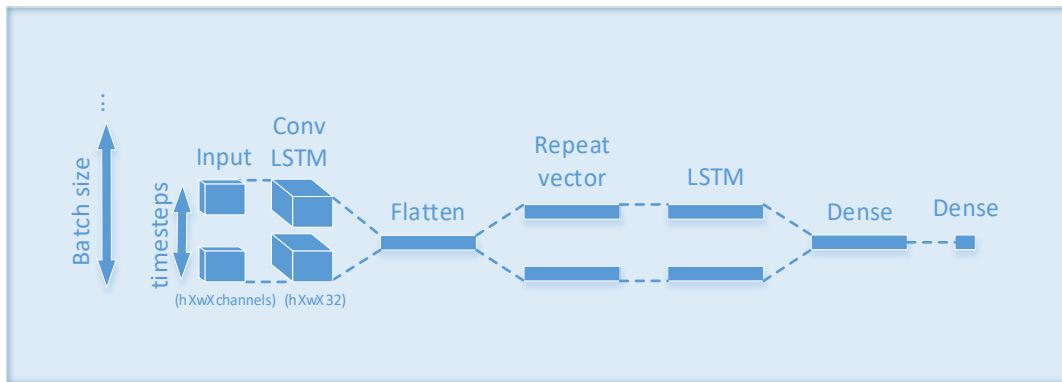


Figure 37: The ConvLSTM model network structure created, own elaboration

The shape of the input for a convolutional LSTM layer is shown below. It uses the channels, rows and columns as needed to correctly apply the filters and being able to calculate the size of the picture matrix.

(samples, time steps, channels, rows, cols)

Table 26: Layers size output of the ConvLSTM NN, own elaboration

Layer type	Output shape
conv_lst_m2d_1 (ConvLSTM2D)	(None, 1, 254, 32)
dropout_1 (Dropout)	(None, 1, 254, 32)
flatten_1 (Flatten)	(None, 8128)
repeat_vector_1 (RepeatVector)	(None, 10, 8128)
lstm_1 (LSTM)	(None, 32)
leaky_re_lu_1 (LeakyReLU)	(None, 32)
dense_1 (Dense)	(None, 32)
leaky_re_lu_2 (LeakyReLU)	(None, 32)
dense_2 (Dense)	(None, 7)

Table 26 shows the size of data passing through the NN for a batch size of 32, 10 time steps and 7 classes. A first convolutional LSTM layer is followed to a dropout to avoid overfitting. In the next

step it is necessary to prepare the data fed to the LSTM. First, a flatten layer is used to give the shape required to store the features of the LSTM. A repeat vector that adds the dimension used by the time steps is added after and therefore the LSTM layer can be used. To include the activation function leaky ReLU and apply it to the model, it is necessary to create an own layer for it. Regarding the following dense layers, the first has the shape of the batch size and the second of the quantity of classes. The code below shows how to create this code with Keras:

```
model = Sequential()
    model.add(ConvLSTM2D(filters=32, kernel_size=(1,3), activation='relu',
input_shape=(timesteps, 1, patch_window[0], patch_window[1])))
    model.add(Dropout(0.5))
    model.add(Flatten())
    model.add(RepeatVector(outputsize))
    model.add(LSTM(length, return_sequences=False))
    model.add(LeakyReLU(alpha=0.05))
    model.add(Dense(length))
    model.add(LeakyReLU(alpha=0.05))
    model.add(Dense(outputsize, activation='softmax'))
```

### 11.2.2 ModelType Enumerator

*ModelType* is a custom class that contains an enumerator naming the possible models. It helps organizing the processes adapted to the model that is applied. Used to sort code regarding the model type to work with, it, for example, determines the requirements for the selected model to train, or it organizes the data accordingly. The training classes, the generator and the final system detector have a *ModelType* object. The UML of this enumerator can be seen in the annex at the Figure 85.

### 11.2.3 Trainer

The code used to train the NN is differenced in two parts, first a parent class called *KerasTrainer*, and second the trainers that inherit from this general trainer. This way it the code can be easily escalated. In the annex Figure 85 is the UML used to design the trainer structure.

#### 11.2.3.1 *KerasTrainer*

This code is complex, and the constructor declares a wide amount of class variables, between them are included, paths to reach datasets for training, validating and testing or a model path in case it is wanted to load one, also variables to determine the size such as the patch window which is a tuple that contains the size that the images are rescaled to, the number of labels, the quantity of epochs, the batch size or the number of time steps. Regarding training settings there is a variable to define the optimizer and also the early stopping and the model check point that are explained below to understand their applicability.

In first place, the variable *early\_stopping* calls the method *EarlyStopping* that inherits from *Callback* class in the Keras library, it stops training when a monitored quantity of progress stops, the patience parameter determines after what amount of times that the model has not improved, it will stop training. The monitored value to determine if the model is learning is the loss as it is set in the monitor parameter. In second place, *model\_checkpoint* is calling *ModelCheckpoint* from the Keras *Callback* class, it saves the model after each epoch, as the parameter *save\_best\_only* is set to true, it will only save the best model until the moment, allowing to have at each training the best result obtained saved. The model is saved in an *.hdf5* format making easier to load it after.

Regarding the class methods, there are some general ones to be used by the classes that inherit from this class. Their applicability is going to be explained below.

### **Multi GPU training**

For parallel training is needed to set the model as a multi GPU model using the function *multi\_gpu\_model* from TensorFlow that receives the model and the number of GPUs. This function is deprecated and is going to be removed after April of 2020, it is necessary to change it for *tf.distribute.MirroredStrategy*.

To see all the devices of the system, including CPU, the function *get\_available\_devices* returns an array with the name of the devices that can be used for training, this list is accessed using the method *list\_local\_devices* from the *device\_lib*, this method returns a list of *DeviceAttributes* protocol buffer, and is from this buffer that the list of names is taken from. To get the concrete number of GPUs in the system is used *show\_gpus*, which is a function that dynamically gets the number of GPUs that are integrated in the system, it prints the result of the function above described. After this, it calls the function *contrib.eager.num\_gpus* from the library TensorFlow, this function returns the number of disponible GPU devices. In the code below is

```
try:
    print("num devices: ", _get_available_devices())
    num_gpus = tf.contrib.eager.num_gpus()
    print("num gpus: ", num_gpus)
    model = tfk.utils.multi_gpu_model(model, gpus=num_gpus)
    print("multi gpu")
```

### **Channels Order**

This method returns a tuple that has the correct shape regarding the format that the data will have to follow training depending on the current Keras data format, whether it is channels first or channels last. When the data format is channels first the Keras input for image processing follows is organized as a tuple of the shape (1, height, width) if it is the other case possible, the order is (height, width, 1).

## **Evaluation**

The evaluation is done over the test dataset, that have not been used during the training, to see the abstraction to new data capability of the model. To evaluate the model there are different methods from the Keras library used.

First, using the *predict\_generator* method, that generates predictions from input samples that are taken from a data generator, in this case, using the custom generator created. It returns a NumPy array of predictions.

Second, using *evaluate\_generator*, this method evaluates the model using a generator, it receives both the test input and the output expected. It makes predictions and then evaluates the performance of them comparing the results obtained with the ones expected. It returns a list of scalars, in case of the model having multiple outputs, or a scalar test loss in the case of this code as there is only an output in the model.

Third, using the *predict\_classes* method that generates predictions for the samples batch by batch, a difference with the previous two evaluation system to take into consideration is that this method is loading the test data fully into memory instead of using a given generator.

A last method saves the training and loss accuracy using the matplotlib library to generate the graphics.

### **11.2.3.2 Trainers**

The classes *TrainLeNet5*, *TrainAlexNet*, *TrainLSTM* and *TrainConvLSTM* inherit from *KerasTrainer*, and the constructor calls the parent constructor to initialize its variables, it also introduces dimension variables to work with RGB or greyscale channels and have a *ModelType* object to determine the input structure required for the different NN.

## **Fit**

Their method *train\_fit* loads the model and trains having all the data loaded in memory simultaneously. The variables data and labels store the training set and class data into a NumPy array.

The code then is split into train and test, groups, they are used during training on each epoch, the first group to train and the second to validate the results. All the arrays of data are reshaped then to adapt to the model input. The model is loaded on the device CPU to leave free space to the GPU to make the training calculations and finally the model is fit using the fit function. In the next code is shown how to ensure using TensorFlow that the model is loaded in the CPU.

```
with tf.device('/cpu:0'):  
    model = get_X_model(...)  
history = model.Fit(...)
```

The *fit* function receives as parameters the training data and labels, the validation data that is used after each epoch to check with a group of it has not being trained with during the epoch. It also receives the batch size, so the fit function can know how big are the bunches that each epoch must divide the data in. Also the numbers of epochs, or full iterations over the training set, the verbose that is set to 1, it means that during the training there is information about the process on the console, that can be of great help to visually evaluate the speed of the training and learning. There are also the *early\_stopping* and the *model\_checkpoint* callbacks between the parameters. This *fit* function will update the weights of the model during the training and will return the final model with its weight.

### **Fit Generator**

The *method train\_fit\_generator* is similar to the *train\_fit* method with the main difference that it uses a custom generator to load in chunks the data with the structure required for the model. It also uses StratifiedKFold from the library Scikit-learn to make cross-validation with 10 folds.

#### 11.2.4 Custom data generators

The class *DataGenerator* contains the custom code of a generator that feeds the data in bunches to be fitted for training. Its UML design can be seen at Figure 85. This class inherits from *Keras.utils.sequence*, which is a base object used for fitting into a sequence of data. To use this base object, it is needed to implement the functions *\_\_len\_\_* and *\_\_getitem\_\_*, and in case there is the need to modify the dataset between epochs it is required also to implement *on\_epoch\_end*.

*\_\_init\_\_* is the constructor that initializes the object, as arguments to initialize its values it receives the following data: a dictionary containing the physical addresses of all the frames and their labels. This way it is possible to access all images in an ordered mode. This constructor also receives the batch size that is used to determine the quantity of elements in a cluster that are taken, to be processed as a group.

The value *dim* represents the tuple that determines the dimension of one element. The constructor also receives the time steps that are required during training, and it will also affect to the dimension of the elements as the LSTM and ConvLSTM NNs will use this feature.

A variable that contain the number of classes is used to transform the output data to categorical and then be able to compare the distributions with the real ones when training. It works as follows, if there are three labels and one element of the dataset that is going to be used for training contains the first label, the distribution of its expected output is (1,0,0).

Also the path where images are wanted to be load from in batches is set on the constructor, the patch window, that contains the height and weight to resize the image, a *ModelType* object and finally a Boolean to determine if *on\_epoch\_end* is applied a shuffle.

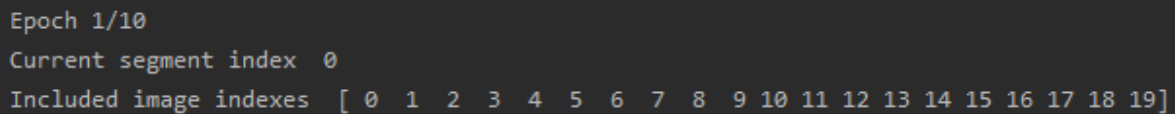
The function `__len__` returns the number of batches per epoch. This value is typically calculated as the number of samples divided by the batch size, this way can be analysed the maximum quantity possible of images without repetitions of the same image on each epoch. The quantity of samples value is taken from the length of the dictionary that contain the frames addresses.

As an example of this, with 31 samples and batches of ten images, the data is distributed as “31/10 = 3,1”, in the code this result is parsed to integer because the batches that are being taken can only be a natural number or 0, therefore the result will be 3 batches per epoch.

```
def __len__(self):
    return int(np.floor(len(self.list_IDs) / self.batch_size))
```

The function `__getitem__` overrides the same method from the inherited Sequence class. The current position in the dataset is controlled with an array of indexes, the indexes is a class variable from Sequence that is used to point to a different group of elements on each batch avoiding repetitions.

These indexes have been used to check the distribution of the elements to be feed to the trainer. In the Figure 38 can be seen the indexes in which a dataset is divided into. Each index references to a position of the dataset, for example the index 0 references to the first image, let’s say position  $x$ , then the index 1 references to the image at the position  $x+n$  and the index 2 to the one at position  $x+2n$ .



```
Epoch 1/10
Current segment index 0
Included image indexes [ 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]
```

Figure 38: Screenshot that shows the indexes that point to the batches made for the training, taken during a program run

It uses the custom method `__data_generation__`, where the indexes are used to iterate over the dictionary of images and get all the data needed to conform each of the batches. Finally, returns a batch of data to be used during the training that consist of tuples of image data and their corresponding label. The shape of the images depends on the *ModelType* enumerator used in the class.

`__data_generation__` is a custom method created to generate data; it is called by `__getitem__` receiving an array that contains the name of the elements to get on each batch and an empty array “X” which shape depends on the *ModelType*. It returns two NumPy arrays, one called “y” contains the labels of the elements in the batch, the other, “X” filled with the data from the images. First, it is created an empty NumPy array for storing the labels, “y” with the size of the batch and volume needed to store int values.

The data is captured, iterating over a for loop, that takes every segment of the array `list_IDs_temp`, that contains the segment of the dictionary with the names of the frames that have to be captured for the current batch. Each image is read using `imread` function from the public OpenCV library and

resized according to the required criteria, after editing the image, it is added to the “X” array and from the name of the image is taken the label value and stored on “y”. *on\_epoch\_end* is the last method of the class. On it, the indexes of the batches are updated at the end of each epoch. If the shuffle variable is set as true, the values are returned in a random order.

### 11.2.5 Other Generators

Two alternative generators have been created; their UML can be seen at Figure 89. *DataGeneratorOfSection* is a class that works as the custom *DataGenerator*, but it also receives the indices created by StratifiedKfold to apply cross-validation. Instead of generating data of the whole dataset, it does it for the indexes given. It is called inside of an iterator of each split for the cross-validation, on the trainer class.

The *DataStratifiedGenerator* is another generator created. It is designed to make the batches for cross-validation without the need to use the StratifiedKfold from Scikit-learn

It is like the custom *DataGenerator*, but on the constructor it is include the counter called *batch\_count* that is initialized to 0 and increases and the *validation\_count*, that it is set to the division value -1, it decreases one on each epoch and determines which batch will not be taken on that epoch, since is going to be the validation data.

Finally, it also has a Boolean to determine if the generator is created for the training or the validation, depending on it, will create batches for the training the validation. To visualize with an example, if there are 3 divisions, on the first epoch there are two batches for training and one for validating. The possible index values for *batch\_count* and *validation\_count* is {0,1,2}, these values are the indexes for the *\_\_getitem\_\_* method. In the first epoch, *batch\_count* will have the values 0 and 1 and the *validation\_count* 2. In the second epoch, *batch\_count* will have the values 0 and 2 and the *validation\_count* 1. And, in the last epoch *batch\_count* will have the values 1 and 2 and the *validation\_count* 0.

### 11.2.6 Frames Provider

The class *FramesProvider* manages the loading of frames into memory. The UML design is reachable in the annex Figure 85. The constructor receives the necessary variables for load and then organize the image shapes, the directory of frames, such as the batch size that is used for training the images, the time steps, the dimensions of the image and the patch window or the model type.

There is a method in this class that is used to return all the frames from a directory as a NumPy array in the correct structure format depending to be input of a traditional image processing NNs with channels last, a LSTM or a ConvLSTM layer.

The elements are read from the directory where the frames are stored using *listdir* function from os library, and these elements are sorted to be taken following a decimal order and iterated with a for

loop. Each image then is read using *imread* function from the OpenCV library, then if the image is correctly read it is resized using again the OpenCV library, to fit into the patch window shape, that gives all the images a fixed size, independently of their original size, so all the images that are used during training have the same size. Finally, the data gets structured in the pertinent way. This code is used when training with a fit and for testing, but with small groups of data, as it loads into memory the complete dataset.

*Get\_training\_set\_and\_class* is a method that receives a patch window and using the path of the folder where the frames are stored, returns two arrays, the first containing the frames and the second containing the labels. It iterates over the list of directories and sorts the elements so the frames that have been stored being classified by their name ordered following the decimal system, are taken on that same order. Each image is read and resized to the given patch window is added to the array of frames data, and the label of that frame, which information is written on the name of the frame file, is stored on the labels array. Finally, once all frames and labels have been loaded in the two arrays, these arrays are structured in the function structure data and finally returned. This code is only used when the training is done using fit during training, as it loads into memory the whole amount of data that is going to be used as training set. Due to the big amount of data it was impossible to use it with the resources of the computer used for training, therefore this code is not being used and instead the data is being loaded on chunks using a custom generator.

*structure\_data* gives a structure to the frames and its classes that is adapted to the requirements for typical image CNN model to the dataset and to the labels arrays, converting the arrays to NumPy arrays and giving to each element on the frames array 3 dimensions, that are (height, width, channels).

### 11.2.7 List ID Dictionary

The *ListID\_Dictionary* contains a dictionary used to facilitate the access to the dataset elements in an ordered way without having to load them all at the same time. This way is ensuring that the time steps order is kept. In the next paragraphs are explained the details of the two different dictionaries creators implemented. The UML design for the code to control the dictionary can be found at the annex Figure 85.

The first one, called *path\_file\_dictionary*, receives as argument a path name and generates two dictionaries that are returned. The first will contain, ordered, the names of the frames that are contained in the path, and the second will contain the extracted labels of the frames on the same order. The code opens the path, which is a dataset folder and iterates through it catching the image names and saving them in the dictionary as follows.

```
{0: '0_x_.jpg', 1: '1_x_.jpg', 2: '2_x_.jpg', ..., n: 'n_x_.jpg'}
```



It is needed to specifically sort the images as their correct time order follow the structure seen in the code above, where the number before the first “\_” symbol represents the step position of the frame in a decimal base and the value after it represents the label for that image. The List of directories on the given path are taken using *listdir(path)* and a for loop that takes the elements sorted. When taking the images without sorting them first, the image “10\_x.jpg” is considered to be before the “2\_x.jpg”, because the first value on the first string name is lower than the one in the second. It is also used a lambda function, so the sorting criteria doesn’t include the values after the “\_” symbol.

The second dictionary generator, *random\_part\_dictionary*, receives two dictionaries, one with frame IDs and the second with label IDs and also a numerical value that determines the size of a segment that wants to be taken from the dictionaries. The elements taken respect the time order. With the random function it is selected a starting point between the beginning of the dictionary’s length, and their longitude – the desired longitude of the segment, so the segment is not out of range. Then, a dictionary that contains the elements from the segment from the random point and the length received by parameter is returned.

### 11.2.8 Video Provider

The class *VideoProvider* is the parent class that is used to manage the datasets. The UML structure of the code is available in annex Figure 86. This class contains code to manage the organization and storage of the video frames, respecting the order and setting the frames labels. The classes *VideoProviderIxmas* and *VideoProviderUT* inherit from the video provider and adapt the frame organization depending on the dataset where it comes from, the IXMAS or the UT dataset.

The constructor receives the addresses of the data that is going to be processed, origins and destinations. It also initializes a counter used to determine the current frame numeration. And finally, is also initialized the number of elements that might be removed from the beginning and the end of the video to shorten it.

The classes *VideoProviderIxmas* and *VideoProviderUT* offer modules to return lists containing frames and their classes from a dataset, to return the entire dataset and the labels. But also, to read all videos from folders and subdirectories and save their frames in order. They bring methods that can manage the datasets removing the x first and last frames of each video in the generated frames dataset, or, to relabel the first and last x frames to a standing or non-violence label. They also can relabel the IXMAS and UT dataset to binary, violence/non-violence. For the IXMAS dataset it also gives the opportunity to separate the videos in 5 datasets depending on the camera position of the video.

For this class implementation, the library OpenCV is of great use to process the videos, specially class functions such as *videoCapture()* to read from video, *imwrite()* to save the frames, *release()* to

close the file, `destroyAllWindows()` to close all windows and de-allocate the memory usage, to resize or to change colour format.

### 11.3 Alarm System

In this project it is included an alarm system that operates in a real time video classifier which UML design is found in the annex Figure 87 and Figure 88. It reads and analyses real time videos and in case it notices violent content a sonic alarm will operate. This software is a simple console application, its menu is shown below at Figure 39.

```
WELCOME to the video Violence Detection Alert System, please choose one option
Enter option:
(t) = Train, (r) = Run Alert System, (e) = exit
```

Figure 39: Console application menu for executing the alarm system, screenshot of the program run

When running the alarm system, the trained model is loaded, and it is used to evaluate the video surveillance cameras output. For this the frames of the video are loaded using the OpenCV library.

The system is prepared to operate with image processing NNs, LSTM and ConvLSTM, the dimensions of the data input for the network is controlled with the *ModelType* enumerator.

The frames processed appear and disappear in order in a fixed window created using the library OpenCV, therefore to the human eye it looks like a video. The frames are accumulated on small batches of 20 frames of data that are feed to the Keras evaluate function and the output is processed.

To avoid fake alarms, the system has a sensitivity of  $\frac{1}{2}(\text{batch\_size})$ . Which means that one half of the processed frames on each epoch must be considered as containing violence by the machine to make the alarm sound and save the frames that contain the event.

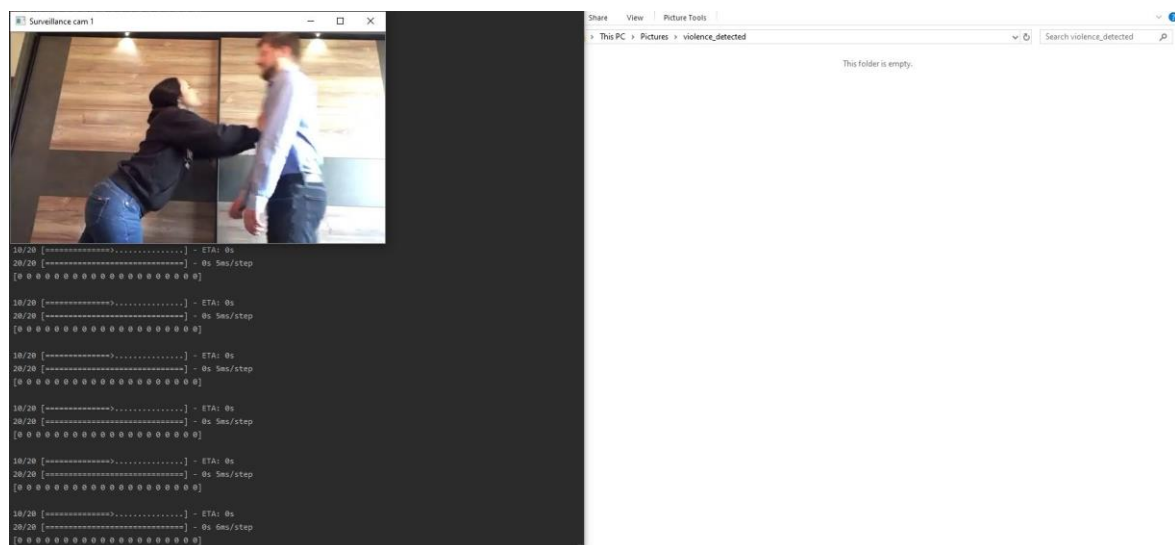


Figure 40: Alarm System running during a violent situation, screenshot of the program run

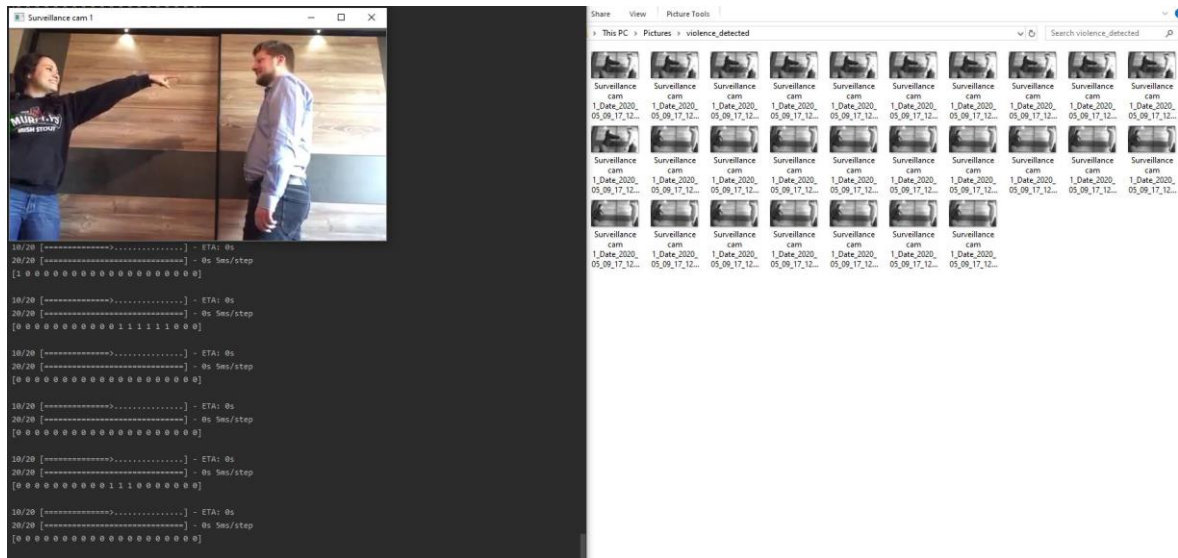


Figure 41: Alarm System running after a violent situation, screenshot of the program run

As an example of how this system works, is shown above two screenshots taken while the execution is taking place. Figure 40 and the Figure 41 show the alarm system software running: Figure 40 shows the system at the beginning, while the Figure 41 shows the result directly after the violence scene. After the situation, one person pushing the other, the frames related with violence are saved on the system in black and white, including camera name, date and time of the happening.

## 12 Experimentation and Discussion

Regarding the investigations and suppositions exposed above, there is a big amount of trials done to support and evaluate them and to find the most suitable result. In this chapter are the main experimentations described and discussed. The full set of trainings is not listed here since this would go beyond the scope of this study. The focus is set on what are the most interesting discoveries and results. To ensure an optimized output of the project, the experimental section is aligned closely to the PDCA idea of EN ISO 9001 (4). During the process of creating code and training the model, checks are done to analyse the current state and to develop and apply correction measures.

### 12.1 Training Comparative Studies

In first place, the modifications made on the datasets. In second place, the training of CNNs: the custom LeNet and AlexNet, training video datasets to determine if a simple image processing NN is capable to solve the problem, using the fit function and loading all the dataset into memory, is checked. All the appearing memory problems are solved to adjust to local training using *fit\_generator* function and a custom generator, which feeds the data in different data sizes, as typically for CNN, LSTM and ConvLSTM input. In third place, the study evaluates time factor dependent NNs for the same problem. There are different approaches to check possible settings. Finally, the two best models are compared, and final tests are run over the alarm system.

### 12.1.1 Datasets

The datasets are subject to different segmentations and modifications in order to search the best option to feed the trainer. In the Table 27 the different subsets created from the modifications made to each dataset are listed. These subsets used for training are explained in the following chapters.

Table 27: Matrix of applied dataset modifications, own elaboration

Mode	IXMAS	UT
<b>Full</b>	Yes	Yes
<b>Section</b>	Yes	Yes
<b>With new standing label</b>	No	Yes
<b>Section binarized</b>	Yes	Yes
<b>Section binarized with new standing label</b>	No	Yes
<b>Full binarized</b>	Yes	Yes
<b>Full binarized with new standing label</b>	No	Yes
<b>Divided by cam positions</b>	Yes	No

#### 12.1.1.1 Full and Section

The training of the full version of IXMAS and the UT datasets results in a big time cost. The time required to train the complete set is calculated and compared using *default\_timer*, from the library *timeit*, because it selects the best clock available for the platform and version of Python.

The time comparison required for training the full datasets is shown for the developed ConvLSTM model using cross validation. This method processes the images maximum once per epoch, with 16 epochs on each of the 10 folds, so in total for 160 total epochs. Table 28 shows the results of the time calculation and the difference between both datasets. The required training time for the full IXMAS dataset is about 10.4 times higher than for the full UT dataset. This big difference is realistic because the IXMAS dataset, with 135079 frames, is about 10.2 times the size of the UT dataset with 13240 frames.

Table 28: Calculation of required training times with UT and IXMAS datasets, own elaboration

Dataset	Training time [s]	Training time [h]
<b>UT (13240 frames)</b>	≈10380	≈3
<b>IXMAS (135079 frames)</b>	≈107677	≈30
<b>Time difference (IXMAS vs. UT):</b>	97297	27

This time effort is not problematic for the one-off final training of the models. But to meet the project time plan, the training time of 30 h must be optimized for the testing in the development phase. The solution is to subdivide the datasets to a reduced size but still varied and big enough to be representative and a good example for learning, and too keeping the equilibrium in the variety of labels. The following distribution for the datasets was applied: The IXMAS dataset (1800 videos) is reduced to 1/10 of the files and approximately 3 hours required training time. For the UT dataset

(120 videos), 3/10 of the files are selected to reduce the training time to about 51 minutes. With this changes and the reduced training time, it is possible have fast and reliable tests. Finally, for the best result, the full versions of the datasets are used to train the finalized models.

*12.1.1.2 Dividing the Dataset of IXMAS on 5 Datasets per Cam Positions*

To check the influence of the camera position, the full IXMAS dataset is used for training dividing the data in 5 groups, as this dataset contains 5 camera positions. When training with these subsets independently is found that the models tend to converge earlier. This is assumed to be because this subset contains less variety of scenes but keeps a big size within the dataset, which makes the models to find earlier common features between the elements. With these results, this dataset distribution is discarded for more use, as it does not bring improvement to the system.

*12.1.1.3 New Standing Label*

After analysing the UT dataset is decided that some modifications on it might bring advantages for training. Approximately the first and last 25 frames per video show the actors standing in front of each other. That might have caused problems for learning as the action standing was labelled with different definitions for the dataset of videos, as a solution it is created a new label for those images.

In the Figure 42 can be seen the starting frames of the first video of the UT dataset that represents a hug.

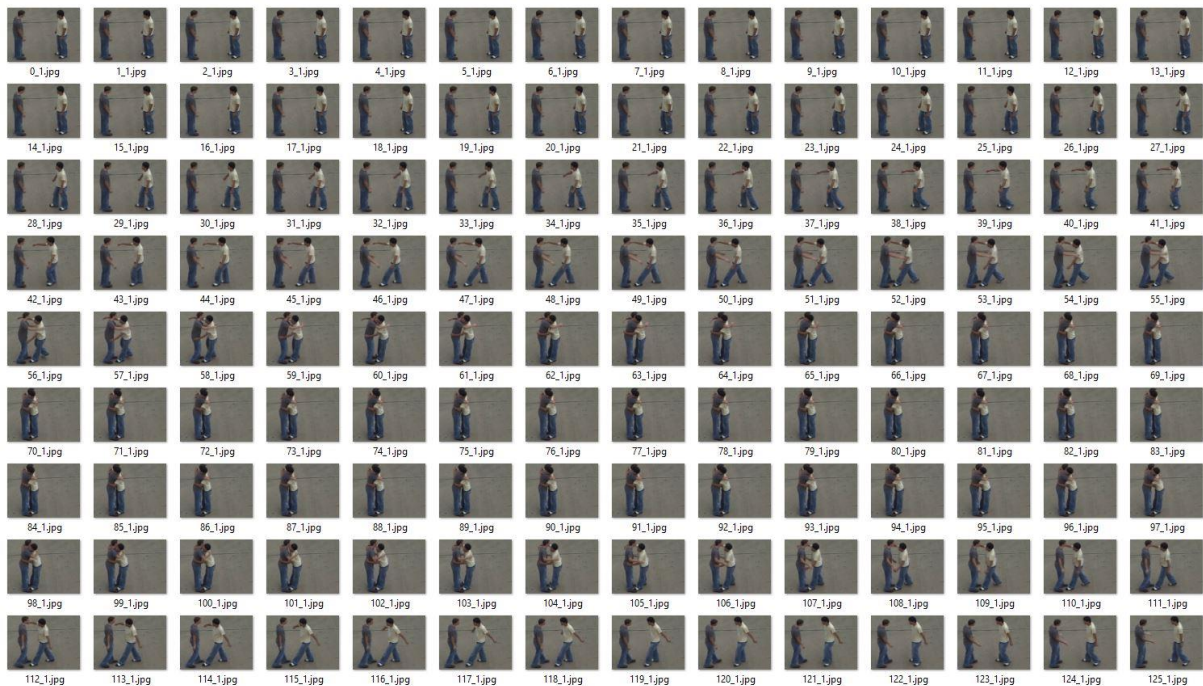


Figure 42: Screenshot of frames from a UT dataset video, extracted from the UT dataset

In Figure 43 is possible to see the first frames of a video in the UT dataset that are labelled as hugging but the action is still not taking place. This segment of the dataset can lead to confusion as every video starts like this regardless of the classification.



Figure 43: Screenshot of frames from the UT dataset that show that there are video frames where there is not the action labelled happening, approximately the first 25 frames of this video express no movement, extracted from the UT dataset

Figure 44 shows frames of a video in the UT dataset hug and it is possible to recognise the action that implies a hug. This segment of the video is considered well classified.



Figure 44: Screenshot of frames from the UT dataset that show that the frames placed on the middle of a video express the labelled action, extracted from the UT dataset

As a simple way to improve the classification on the training process a new label called standing is created for those scenes. This is tested with both datasets and different models. For the IXMAS dataset using this relabel technique did not improve the accuracy, it is expected to be due to there is a small quantity of frames that does not show the action at the beginning and at the end, around 12, less than half of the ones seen in the UT dataset. Relabel the 15 first and last frames of each video, instead of 25 as suggested at the beginning was giving better result. It is thought to be because after transforming the 25 first and last frames from each video, there were too many elements with the standing label in the dataset creating instability between the labels partition when training with binarization as the non-violence label was having more elements than the violence. Therefore, creating the new label or including standing to the non-violence group for the first and last 15 frames for the UT dataset is reducing the mistakes of the machine related with the standing action.

#### 12.1.1.4 Binarize the Datasets

In this project it is tested the datasets with their original labels, with an extra label, but also with only two labels, violence/non-violence. For detection of human violence interaction there is a huge amount of possible actions, and non all of them are being contemplated in this project. Based on the violence detection strategy defined in 10.3, the speed, blur and the space time differences are subject of interest and they can determine violence, this means that violence has a common factor that can be identified in video. Therefore, both IXMAS and UT have been subject of binarization for training.

In Figure 45 and Figure 46 it is shown a representative segment of the IXMAS dataset that was trained with the LSTM model and binarized labels. The results are very good for analysing how the binarization improves the results, even though, 250 epochs are too much for the model ends in overfitting and the validation dataset is unrepresentative and can be seen in the noise of the validation accuracy, this problem was reduced after, applying cross-validation. But as it can be seen the model starts learning at around 88 % of accuracy. This happens when training with less labels, it is more

difficult to make a mistake. And for a binary problem the minimum starting accuracy that was found in the tests was around 50 %.

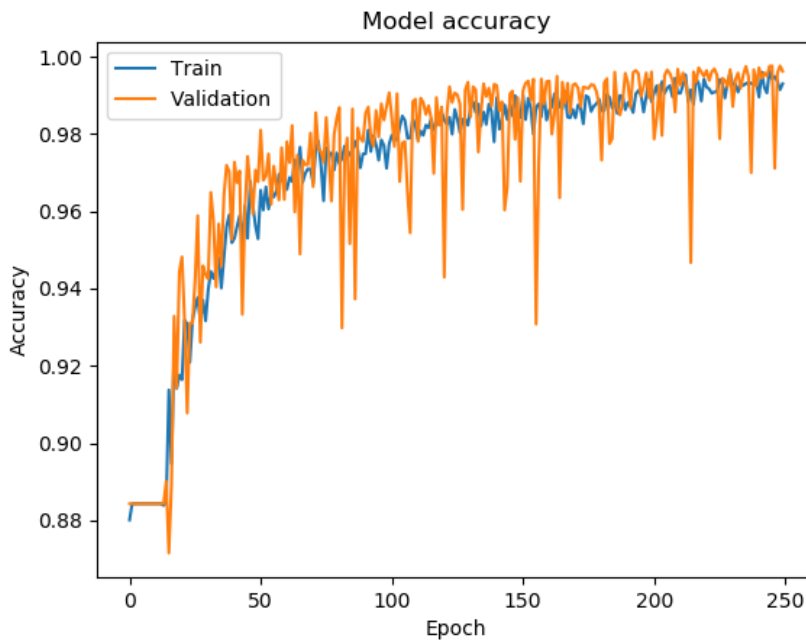


Figure 45: Model accuracy graphic for LSTM with binarized IXMAS dataset, created using matplotlib during the tests made in this project

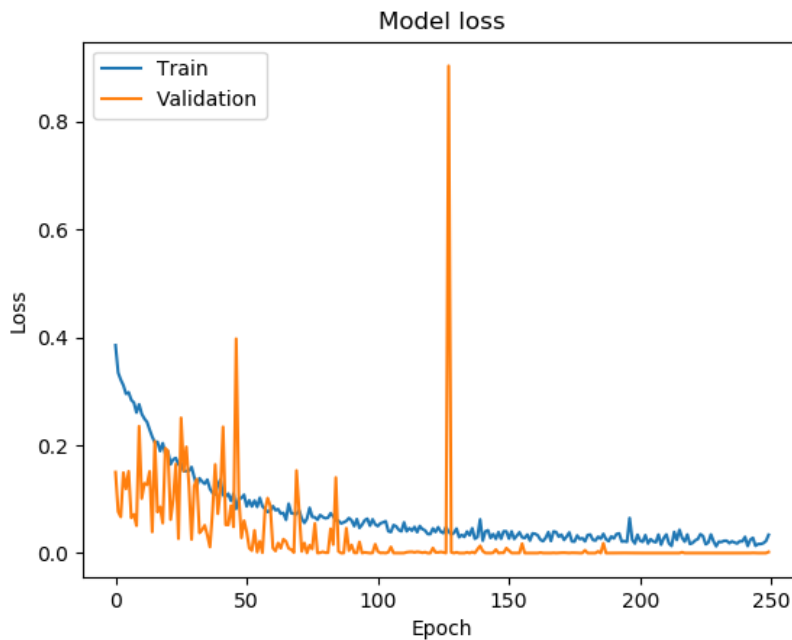


Figure 46: Model loss graphic for LSTM with binarized IXMAS dataset, created using matplotlib during the tests made in this project

As the labels are reduced to violence and non-violence, the model finds common characteristics that represent both labels. Using the original labels, the IXMAS model is able to difference the actions

from each other, but in the binary case small mistakes take place due to the big generalization, this can also explain the noise around Figure 45. The training achieves a good result in accuracy for both training and validation that is seen in Figure 47. Also, when trying the model with the test dataset, the accuracy is very good, except for a few frames that mistakes.

```
356/356 [=====] - 165s 463ms/step - loss: 0.0667 - accuracy: 0.9771 - val_loss: 0.0024 - val_accuracy: 0.9692
Epoch 00250: loss did not improve from 0.06081
[INFO] evaluating network...
Accuracy = 0.8374999761581421
```

Figure 47: Results obtained training a model with a binarized dataset, screenshot from a program run



Figure 48: Mistaken frames while binarization from the IXMAS dataset, extracted from the IXMAS dataset

Figure 48 are the frames that the model mistakes, the model is considering that looking at the clock is violent most likely because it is a similar movement to punch. It is deduced to happen because the model does not have a concrete punch and looking at the clock labels anymore. Therefore, it has not learned the details of those actions independently but mixed with other factors that have the violence and non-violence segments in general. The two actions are similar, and the model makes mistakes in a minor quantity of the frames that represent those actions.

For fixing this problem it is decided to take the steps of signaling the violence alarm after a significant amount of violence prediction is set as output for the model as explained at the chapter 12.2.1. Making the model less sensible, and less prone to small fails. The results are considered positive and that they can bring better options for generalization as there are more kind of actions than the ones labelled on the original datasets, this generalization violence/ non-violence is part of the final product.

### 12.1.2 Image Processing

In first place, are evaluated the two image-based models, one inspired on LeNet5 and the AlexNet. A summary of the results can be seen in Table 29. The complexity of the problem approached in this project resulted into both networks, LeNet5 and AlexNet, are proved to be not of use. AlexNet is capable to learn from images from the UT dataset that it was trained with. It is also successful with ImageNet, or as seen in (27), it achieves an 81,6 % of accuracy on CIFAR-10 dataset. It proves that this NN is good for image recognition, even for the violence in video datasets used, but it is very inaccurate generalizing the test videos in this project. Therefore, it is unsuccessful for human violence detection.



Table 29: Results of the image-based models used in this project, own elaboration

Model	Result
<b>LeNet5 (modified)</b>	Not learning
<b>AlexNet</b>	Good learning but misinterpretation of independent set

The NN based on LeNet5 is not solving the problem and AlexNet has difficulties with the accuracy. The interpretation of the inaccuracy is that they lack important information related to the space-time context. The complexity of human interaction includes factors that an image-based model lacks. For example, in a frame with two people, without the context of how they approached each other, the correct tagging of their action is difficult. The fact that AlexNet is more successful than LeNet5 is considered due to its higher deepness. Both models were considered as an option to avoid the computational cost of including the time dependency used by video processing NNs. This idea was abandoned fast and the image processing CNNs replaced by a LSTM and a ConvLSTM for video processing.

### 12.1.3 Video Processing

The Stacked LSTM is the first time-dependent model trained. The video datasets have been trained using *fit\_generator* to be able to train with the whole dataset loading it into memory by chunks. The first test is done with 10 epochs and a batch size as mini batch gradient of size 20.

The first dataset used for training with fit generator is the UT-interaction dataset (38) that has 6 possible labels, {0,1,2,3,4,5} each one represents one kind of action, explained at Table 14 using a colour format that is used during the training is the RGB, and the images have been dimensioned to 256X256 pixels as determined at 10.5.1. The time step value is set to 5 to learn from the interdependency of close images without keeping memory from images that where processed with a big separation.

The results of the first training are negative, neither loss nor accuracy has progress, this tells that the model is not learning. With the model stopping to learn when it reaches a value around the 20 % of accuracy, it is not too promising. Considering that there are 6 possible labels, the results seem to be random. It is important to find the reason for this situation. In the next sections it is explained different approaches to adjust and fix the problem.

#### 12.1.3.1 Normalizing the Data

The crossentropy loss, used on this approximation, is a minimized loss. It is expected to get better results when training with small values as explained at 9.3.4.2. To normalize the data implies a reduction of the range of possible values. For applying this to the work, in the dataset frames the values that represent each pixel value of the image is normalized, being divided by 10.5.1.

Table 30: Accuracies obtained using data normalization with the IXMAS dataset, own elaboration

Model	Labels	Patch window	Normalization	Dataset	Accuracy [%]
<b>LSTM</b>	11	256X256	1/256	Full IXMAS	35
<b>LSTM</b>	11	256X256	No	Full IXMAS	85

Table 30 shows the results applying and not applying data normalization to the IXMAS dataset when using the LSTM model. Applying this change implies a great success to the learning and makes the LSTM model to be capable to achieve an accuracy over 85 % with the IXMAS dataset, when the accuracy without using this adaptation is the 30 %. The best result is achieved combining normalization with making bigger the height and wide of the frames.

The interpretation for this improvement is that big calculated values make the training more difficult due to the sparsity of the values and the possible tendency to the infinite on the calculations. The division of the data values per 256 reduces this problem because the distances of the jumps of the gradient during training are smaller, but also incrementing the size of the image makes more separation between elements and gives more data to the NN.

### 12.1.3.2 Losses

The use of categorical crossentropy and MSE losses is compared in this section. Categorical crossentropy is used to discriminate in which probability each option is the correct one, MSE on the other hand returns only one possible solution. It is interesting that in the same conditions both losses achieve the same accuracy as it can be seen in the Table 31, but the epochs needed for it is different. with MSE it is required 250 epochs and with categorical crossentropy 200, which is a decrease of 20 % of the epochs required to achieve 80 % of accuracy.

Table 31: Different accuracies obtained using data normalization with the IXMAS dataset, own elaboration

Model	Loss	Learning rate	Dataset	Epochs	Accuracy [%]
<b>LSTM</b>	MSE	$1e^{-3}$	Full IXMAS	250	80
<b>LSTM</b>	Categorical crossentropy	$1e^{-3}$	Full IXMAS	200	80

It is interesting to compare the accuracy progress between the model subject of this comparison using MSE loss at Figure 49 and using crossentropy loss at Figure 70. The progression of the one using crossentropy shows a faster learning at the beginning and converges later, this reflects the results on the Table 31. The comparison between the losses, at Figure 50 and Figure 71, regardless the noise on validation caused by an unrepresentative validation set, the progression shows a better evolution on the categorical crossentropy. The training and validation loss are reduced first faster and at the end it converges. The loss for MSE instead evolves with the predictions on the validation dataset that does not go down with the training loss. The decision is that crossentropy applies better for this

problem, reducing the training time and learning better at the early epochs. Therefore, it is selected as part of the final solution.

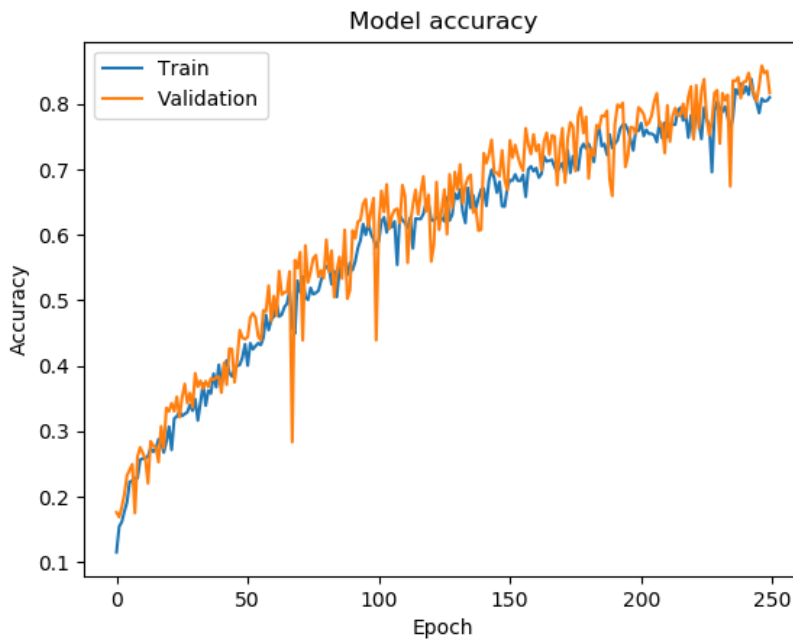


Figure 49: Stacked LSTM model accuracy with MSE loss and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

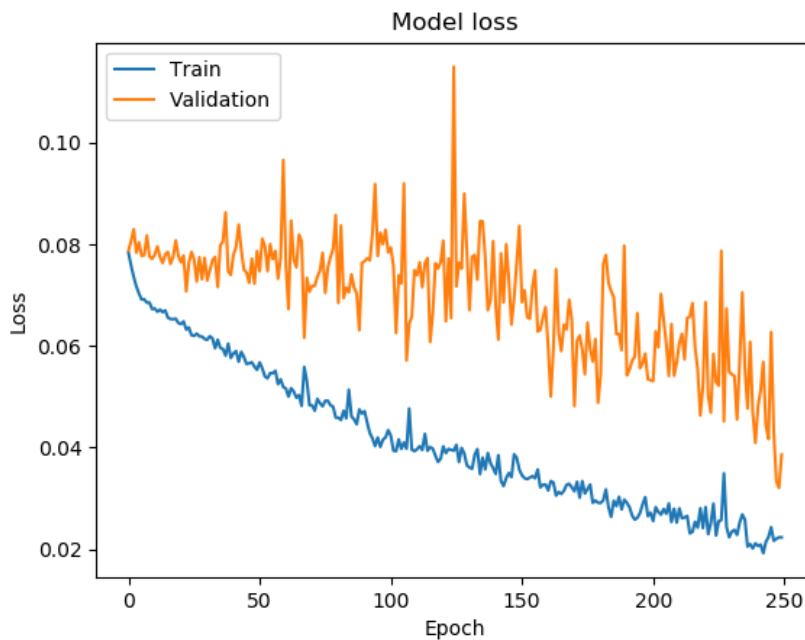


Figure 50: Stacked LSTM model loss with MSE loss and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

### 12.1.3.3 Time Steps Approach

In Figure 51 and Figure 52 the quantity of time steps used is 20, this idea is taken as an alternative to relabel the first and last 25 frames as in 12.1.1.3. Then connecting the info between frames during

the training could reduce this problem. It was also trained with 10 time steps as it is suggested in 10.5.7 the results are shown in Figure 70 and Figure 71.

In the accuracy for 10 time steps, Figure 70, it can be appreciated a better result than in the one using 20 time steps, Figure 52, from 80 % of accuracy to close to 90 %. The validation loss has a better performance following the training loss evolution, which is better than the loss with 20 time steps in Figure 52. This is considered to happen because the training was remembering with 20 time steps irrelevant information that was not found as a common factor in the validation dataset. Then the use of 10 time steps is considered as a better option as it was suggested in the methodology.

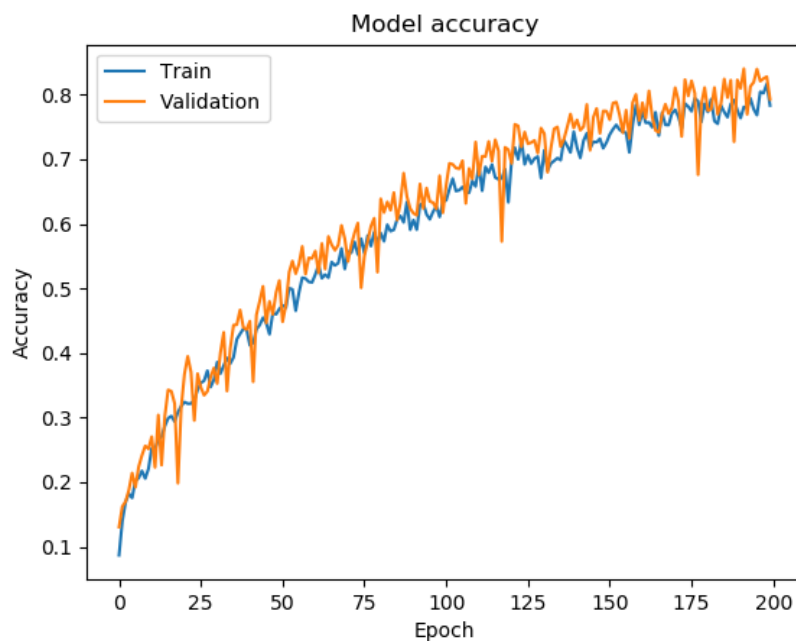


Figure 51: Stacked LSTM model accuracy for 20 time steps and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

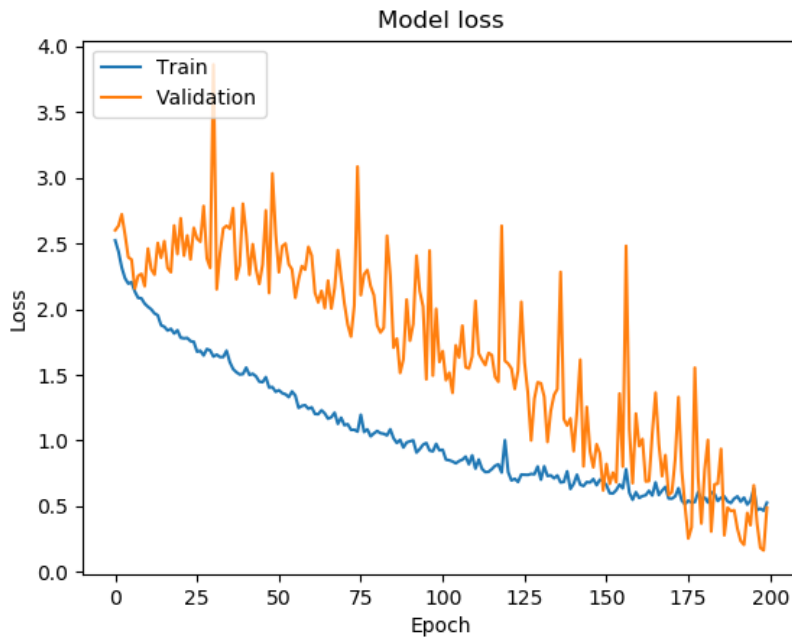


Figure 52: Stacked LSTM model loss for 20 time steps and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

#### 12.1.3.4 Batch Sizes

The batch sizes used for the training are the following {20, 40, 80, 64}. In the tests have been seen that best batch size is between 40 and 80. Using a power of two as recommended at (56) did not seem to cause any improvement during the training.

The stacked LSTM model is mainly trained with a batch size of 40. But as an alternative it has also been trained with a size of 20 and 80 using the mini-batch gradient descend explained and chosen at the chapter 9.3.6.3. In the next graphs can be seen the results for a batch size of 20 and 80.

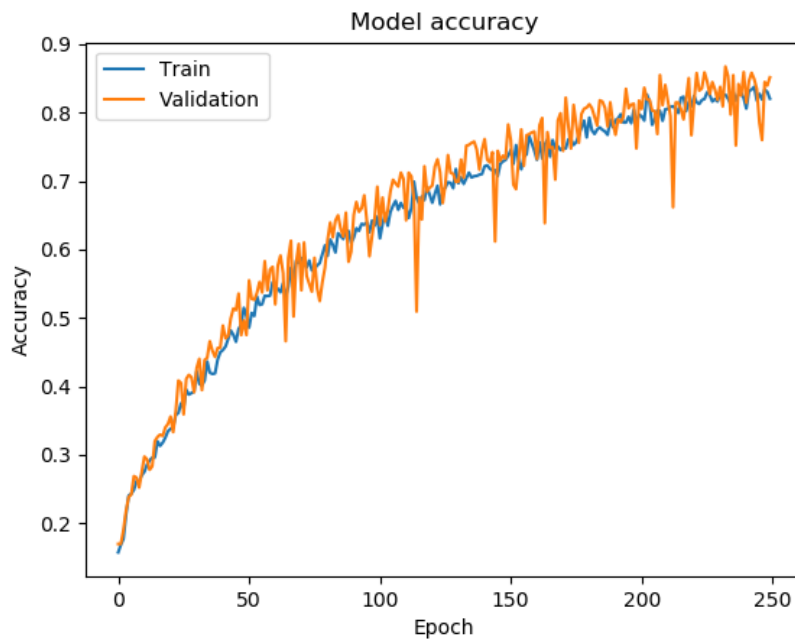


Figure 53: Stacked LSTM model accuracy for 10 time steps, batch size of 20 and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

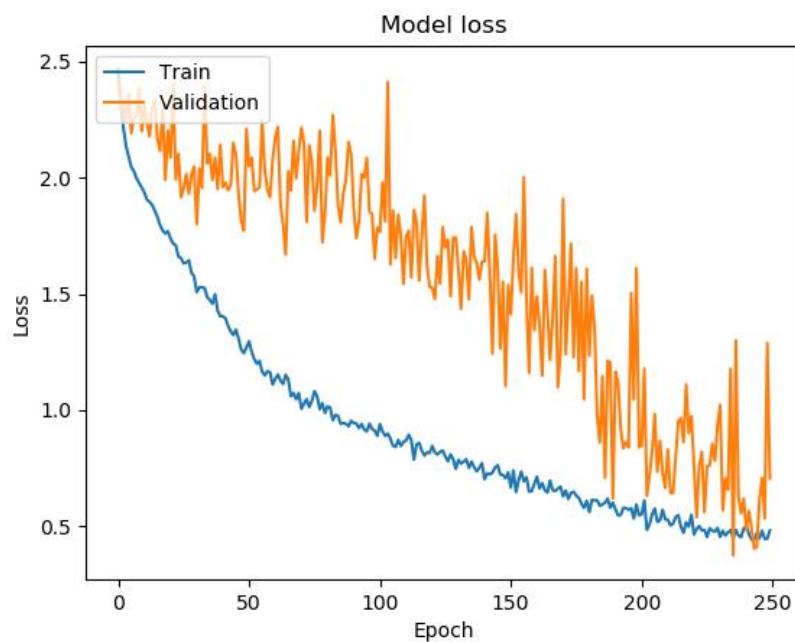


Figure 54: Stacked LSTM model loss for 10 time steps, batch size of 20 and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

As it can be seen in the Figure 54, the model loss of the validation does not evolve as the one of the train, this is telling that the batch size is too small and the validation dataset is not representative. Comparing a batch size of 80 at Figure 55 with one with 20 at Figure 53 in this project, it can be seen how the accuracy for the one with bigger batch size evolves more constant and converges later than the smaller. From the loss perspective, it is also better the results with a batch size of 80, Figure

56, the validation loss evolves as the training without a big gap as it happened with Figure 54. The conclusion is that a bigger batch size, between 40 and 80, brings better results than a smaller one.

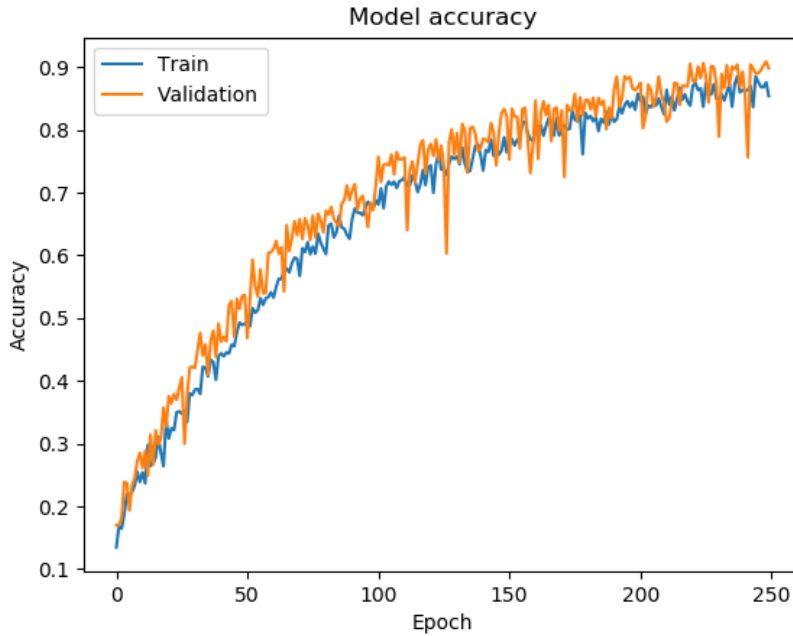


Figure 55: Stacked LSTM model accuracy for 10 time steps, batch size of 80 and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

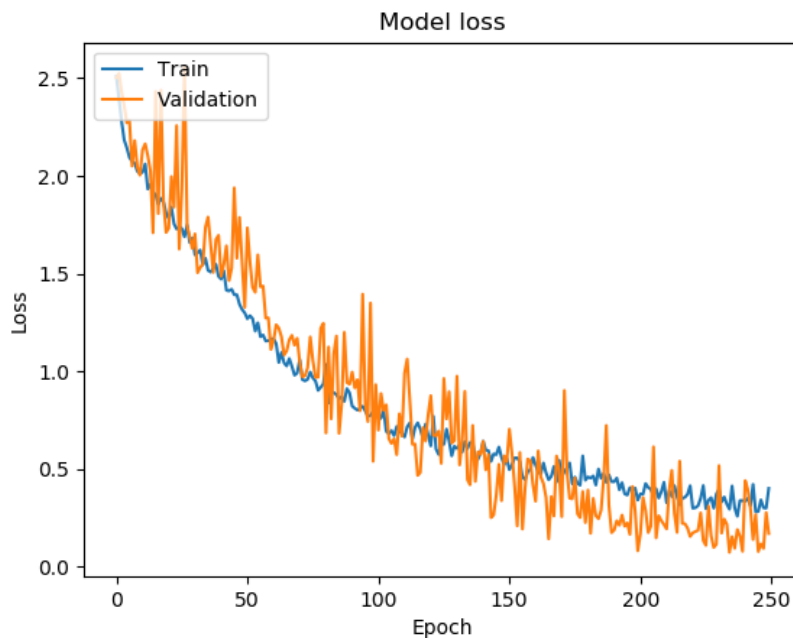


Figure 56: Stacked LSTM model loss for 10 time steps, batch size of 80 and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

### 12.1.3.5 Activation Function Adaptation

One of the early training problems is that the predictions are pushing for only one result when testing the accuracy on different actions. On each training, this final prediction is a different value. The interpretation for this is that the model is not learning properly and therefore not extracting the correct features for correctly predicting a label. And, the weights of the model make to tend only for one label value as result. Analysing the possible reasons for the model to push always for a concrete reason the result is that, it is possible that there is vanishing gradient explosion happening, explained in 9.2.3. The activation function used is sigmoid and as explained in 9.3.3 it can develop in the mentioned problem. To reduce this problem, it is applied normalization, explained at 12.1.3.1, and used ReLU and Leaky ReLU as alternatives to sigmoid. As seen in the Figure 57 the model distributes the probability between the different layers, fixing the previous problem.

The decision regarding the activation function is shown in the chapter 9.3.4.2 where it is seen that the last layer activation that works better with categorical cross entropy is ReLU.

```
Predictions = [0.          0.32202864 0.15701987 0.27770048 0.16311966 0.22292668]
```

Figure 57: Screenshot that shows the predictions for each possible label for a frame from the UT dataset, screenshot from a program run

As seen on the screenshot above the decision taken to predict a label is less hard and different possible labels have more chances to be chosen. This change brought improvement to the performance of the training, but there were still notable during the training tendencies for some labels regardless of the dataset values.

### Leaky ReLU

As explained before ReLU is a good activation functions to reduce the vanishing gradient problem for values over 0 but leaky ReLU also avoids for the negative ones as explained at 9.3.3.2. This activation is applied with two different alphas, first 0.01 and then with 0.05. The two first graphics below show the accuracy and loss result for Leaky ReLU with alpha 0.01 and the two next ones show the result for the alpha 0.05.



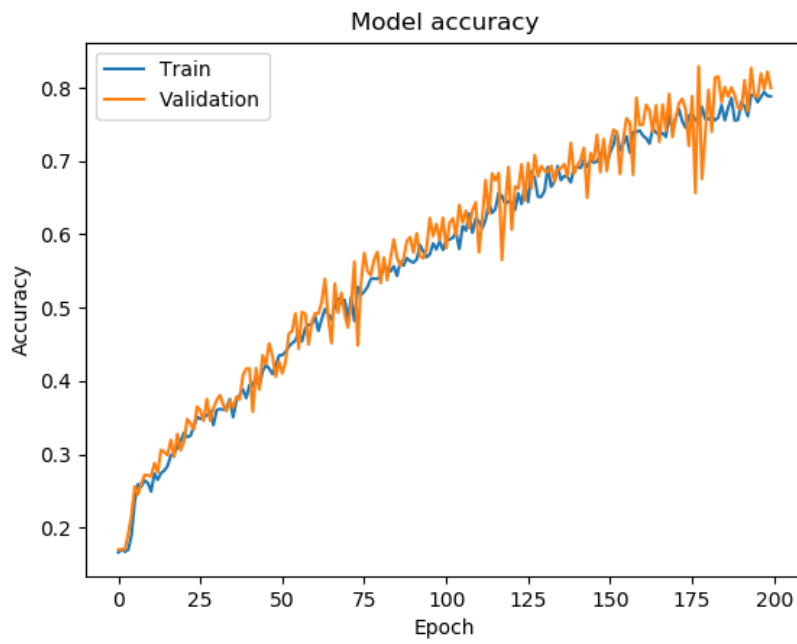


Figure 58: Model accuracy graphic for leaky ReLU model using an alpha of 0.01 with Adamax and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project

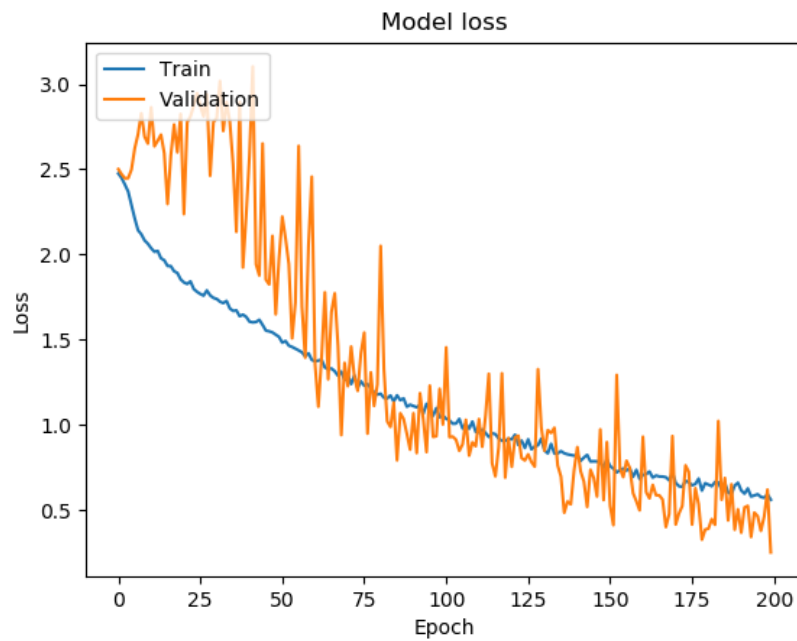


Figure 59: Model loss graphic for leaky ReLU model using an alpha of 0.01 with Adamax and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project

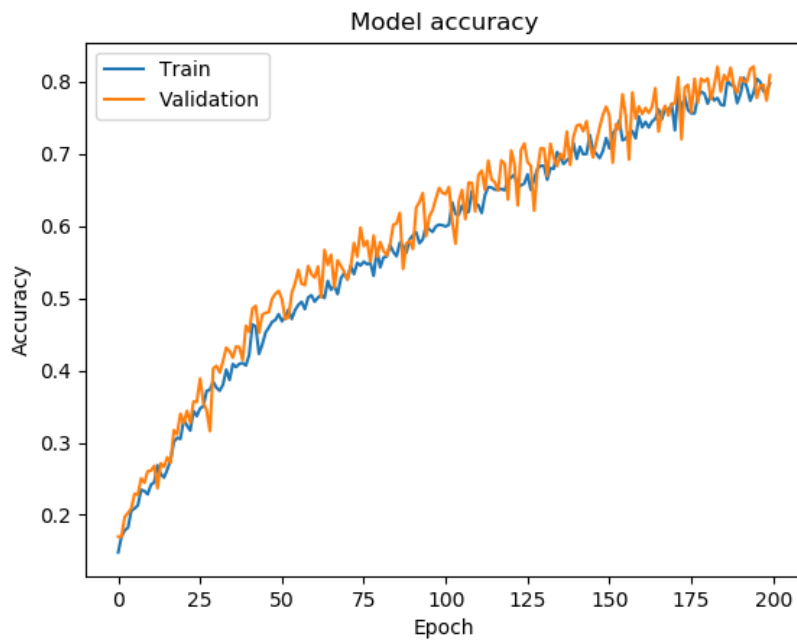


Figure 60: Model accuracy graphic for leaky ReLU model using an alpha of 0.05 with Adamax and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project

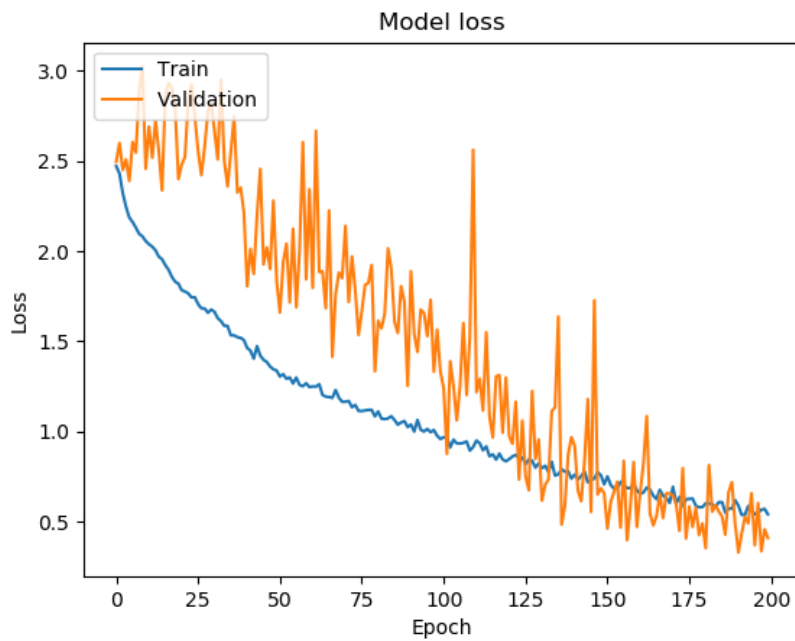


Figure 61: Model loss graphic for leaky ReLU model using an alpha of 0.05 with Adamax and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project

The validation accuracy using an alpha of 0.01 Figure 58 shows more noise than the one using 0.05 Figure 60. The given interpretation is that the one with the bigger alpha can distance more from 0 the negative values giving more negative weight and the training model then is more suitable for the validation images.

On the other hand, for the 0.05 alpha training the loss evolution at Figure 61 shows how the loss of validation takes more time to decrease than the loss of training. Meanwhile in the 0.01 loss results the validation and the training losses evolve in the same way Figure 59.

**Last Layer Activation**

In the investigation to decide the activation function (chapter 10.5.4), the most fitting activation is softmax. In this project is also compared with sigmoid. In Figure 64, Figure 65, Figure 66, and Figure 67 results is confirmed that even when using a less deep NN, softmax activation gives better results, especially when combined with the crossentropy loss as suggested in 9.3.4.2. Therefore, it is confirmed that categorical crossentropy works better with a softmax output as recommended in (31).

*12.1.3.6 Optimizers*

It is interesting the comparison of the results obtained with the two selected optimizers, Adam and Adamax explained at 9.3.2.3 and 9.3.2.6. As an example of their accuracy in this chapter is shown the different outcome of using the two optimizers, with a test using LSTM with leaky ReLU using an alpha of 0,05 and a learning rate of  $1e^{-3}$ . Both optimizers work well for this problem, but there is a detail that is worth to mention: As it can be seen in the Adam graphic results, Figure 62 and Figure 63, the values evolve similar to the ones from the Adamax graphs, Figure 60 and Figure 61. The big difference is the speed that the model needs to achieve a good result level. With Adam the model needs 250 epochs to achieve an accuracy around 95 %, on the other side Adamax it converges earlier and in 200 epochs it reaches 80 % of accuracy this can be seen simplified in Table 32. The reason for these results is estimated to be the norm used for calculating the gradient updates, as this is the big difference between both methods. Therefore, basing the gradient updates to the infinite norm does not give better results than using the square root of the sum of the squares, which is the Adam approximation. The noise in the validation loss was corrected applying cross-validation.

*Table 32: Accuracies obtained using Adam instead of Adamax when the model converges, own elaboration*

Model	Optimizer	Converge	Dataset	Epochs	Accuracy [%]
<b>LSTM</b>	Adam	Yes	IXMAS	250	95
<b>LSTM</b>	Adamax	Yes	IXMAS	200	80

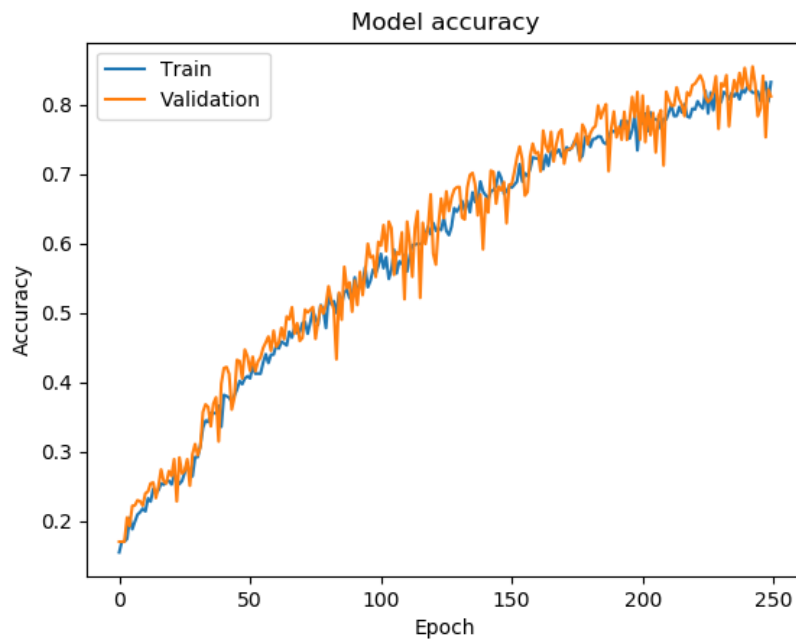


Figure 62: Model accuracy graphic for leaky ReLU model using an alpha of 0.05 with Adam and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project

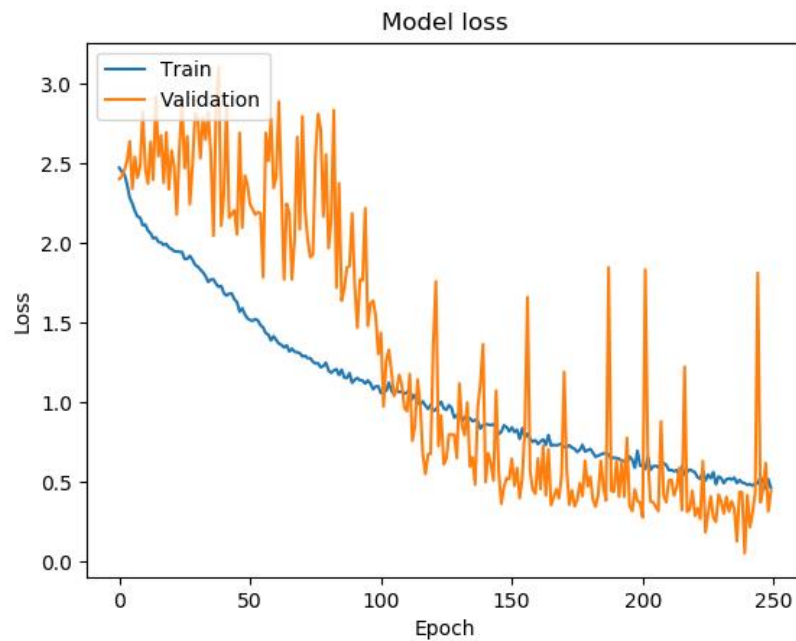


Figure 63: Model loss graphic for leaky ReLU model using an alpha of 0.05 with Adam and  $1e^{-3}$  learning rate, created using matplotlib during the tests made in this project

### 12.1.3.7 Adding Deepness

During the development of the stacked LSTM model development different deepness are tested. To see the influence of the deepness to the result as it is suggested in the 10.3.2. To bring a representation of the deepness application in this project, it is shown the results for the stacked LSTM with two, three, four and five layers of deepness.

On Figure 64 and Figure 65 it is shown the results for five stacked LSTM layers using a final sigmoid activation. The result is still not one of the bests, as other variables have to be adjusted.

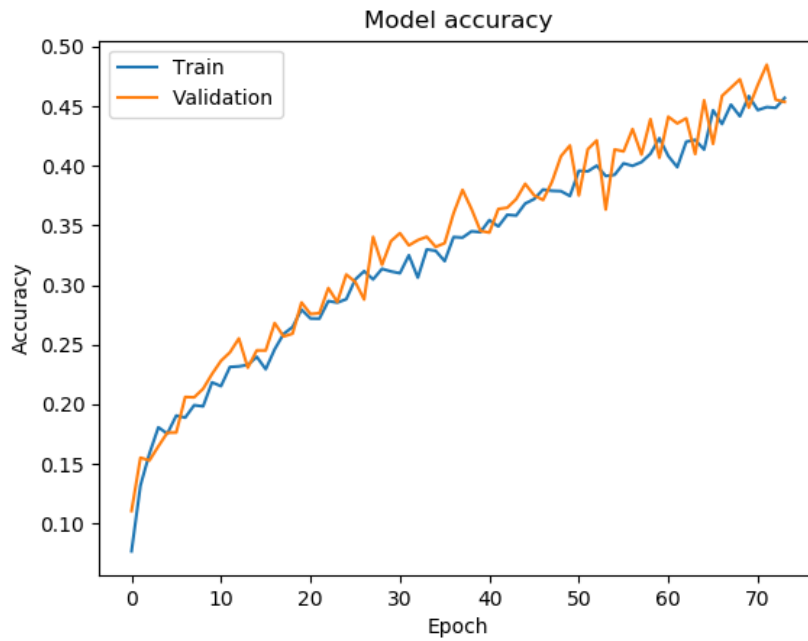


Figure 64: Stacked LSTM model accuracy with five layers of deepness and sigmoid final layer activation, created using matplotlib during the tests made in this project

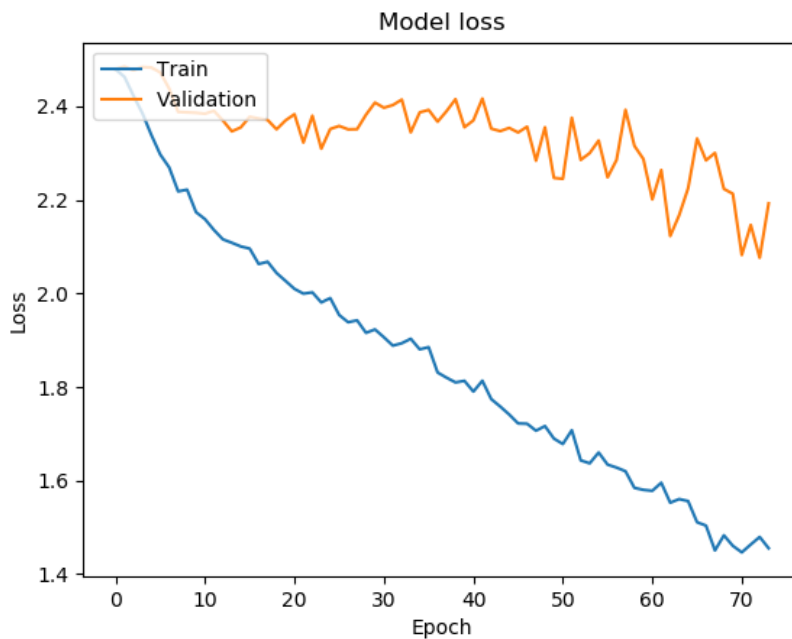


Figure 65: Stacked LSTM model loss with five layers of deepness and sigmoid final layer activation, created using matplotlib during the tests made in this project

Figure 66 and Figure 67 how the results obtained using a stacked LSTM model with three layers with a final softmax activation layer. Bigger deepness does not imply improvements, and the best result

found is the network with four layers of deepness. It is seen on the examples explained before. Probably the deeper networks are learning too many details from the images, which difficult the generalization.

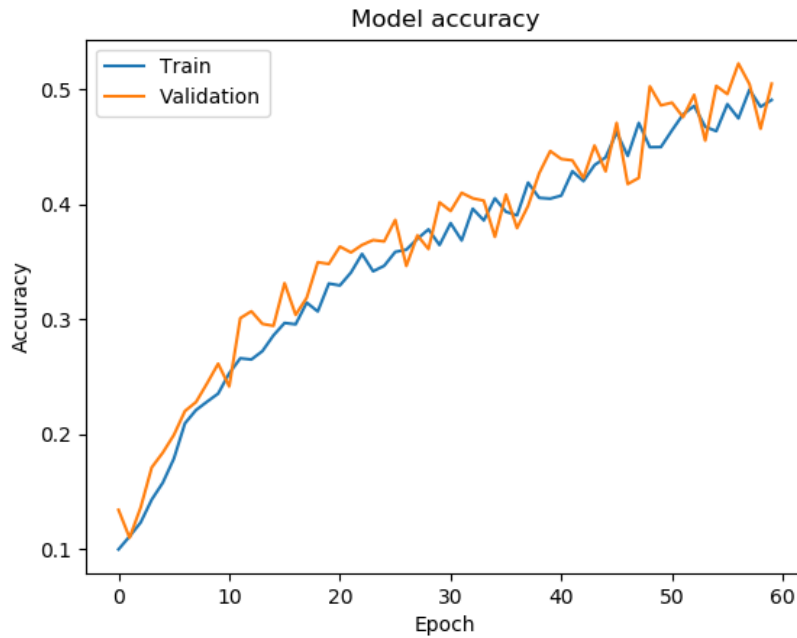


Figure 66: Stacked LSTM model accuracy with three layers of deepness and softmax final layer activation, created using matplotlib during the tests made in this project

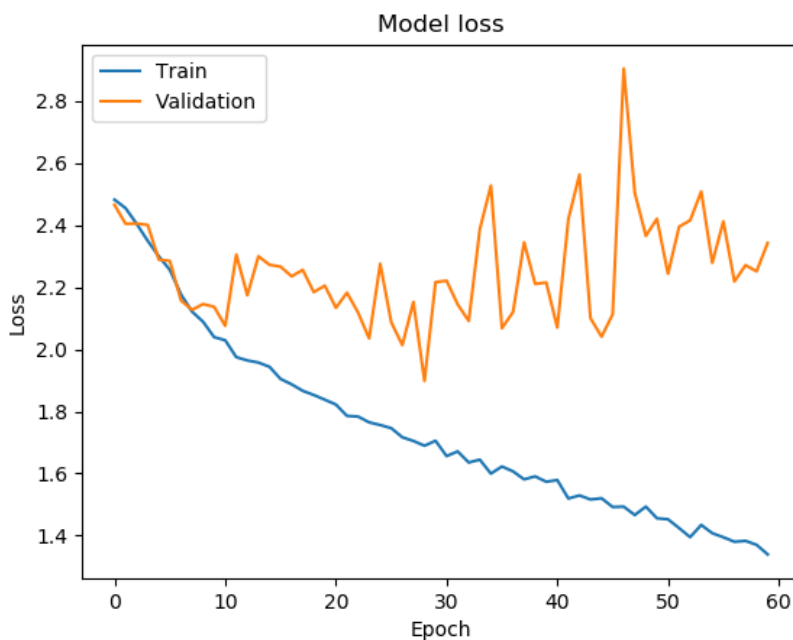


Figure 67: Stacked LSTM model loss with three layers of deepness and softmax final layer activation, created using matplotlib during the tests made in this project

### *12.1.3.8 Changing the Learning Rate*

There are different learning rates that were tried to look for the best speed to learn, the values tried are in the range  $1e^{-2}$  to  $1e^{-5}$ , which is [0.13533528323, 0.00673794699]. The learning rate is chosen looking for a value that permits to learn with a speed that lets the network to train without converging too fast and without being stopped. Reducing the learning rate to  $1e^{-5}$  did not improve the performance of the training.  $1e^{-4}$  was giving better results. One of the best results was with  $1e^{-3}$  that on the epoch 200 could reach an accuracy around 0.80 and still learning but the loss was not improving in 1/4 of the epochs from the epoch 150. Therefore, it can be deduced that the learning rate  $1e^{-4}$  could make the training to train faster without converging too early. And the best learning rate for this problem is proved to be  $1e^{-4}$ , because with the same number of epochs was learning to a good speed.

### 12.1.4 Validations

The validation during training has been done from different approaches, in this section is explained the general results for the different approximations used.

#### *12.1.4.1 Independent validation dataset*

Using the two UT-interaction segments of datasets, one for training and the second for validating, the main difference is that the background is different and so the clothes of the kids. It is done following the decision obtained in 10.5.2.3 where it is talked about how to select a validation style. The first option was to use a non-trained with dataset that had to be similar enough to avoid the problem of validation set unrepresentative explained at 10.5.6.6. the two datasets were not close enough and the graphic showed that the validation set was not representative.

The results were not too promising with the LSTM model and the UT dataset where the validation accuracy was converging when achieving the 57 % of accuracy for 6 labels. The validation set was unrepresentative. This is interpreted as when training with a small dataset, the UT dataset consists in 117 videos, segmenting them into different parts, one for training and another for validating, is not convenient, as it decreases the quantity of trainable elements.

#### *12.1.4.2 Part of Training Validation Dataset*

Using a random part of the training dataset as a validation was giving good results, but as the segment used was also part of the training in the epoch it could lead to overfitting, the overfitting effect is explained at 10.5.6.3. These trainings were successful with the IXMAS dataset achieving an accuracy of 95 % in training and 92 % in validation. It is interesting that there was not overfitting taking place and when testing with a non-trained dataset element, even from the UT dataset the predictions were accurate in a 70 % of the cases, having some errors such as mistaking raising the arm to check the clock with a violent act, most likely mistaking it with a kick action.

For training this dataset it was needed between 150 and 200 epochs. The next approximation explains an optimization applied to reduce this quantity of epochs and give better generalization to the model.

#### 12.1.4.3 Cross-validation

Using cross-validation reduced the quantity of steps needed to achieve a good result from between 150 and 200 epochs to 8 epochs per 10 folds, that makes a total of 80 epochs for the full IXMAS dataset. And as it can be seen in the next two tables Figure 68 and Figure 69 the final results were very positive with a good accuracy for both training and validation.

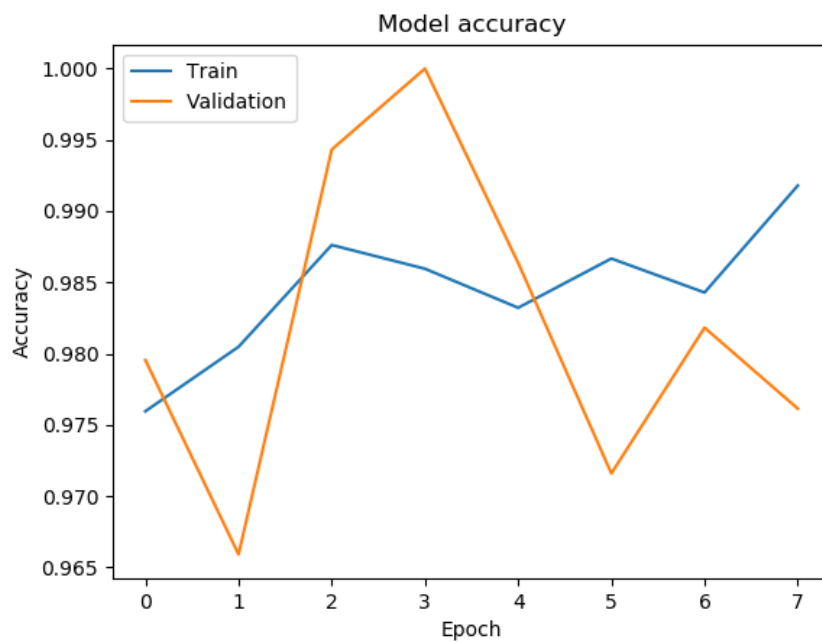


Figure 68: Accuracy in the last fold of a cross-validation using the complete IXMAS dataset and the ConvLSTM model, created using matplotlib during the tests made in this project



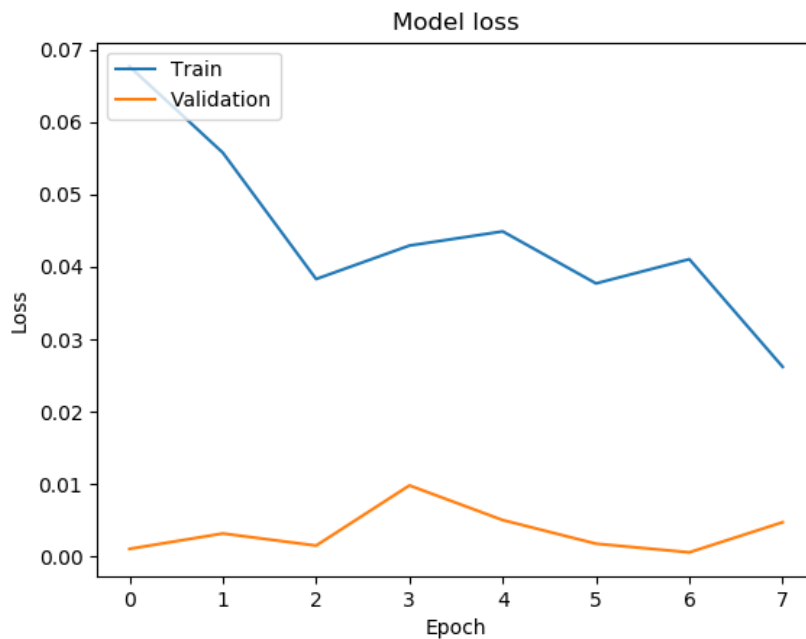


Figure 69: Loss in the last fold of a cross-validation using the complete IXMAS dataset and the ConvLSTM model, created using matplotlib during the tests made in this project

### 12.1.5 Pretrain the Model

One of the proposals introduced in this project is to use one of the violence datasets found as base to train another one, as it was explained in 10.3.4 to avoid overfitting and create a stronger model capable of better generalization.

The ConvLSTM model is Trained using the full IXMAS database achieving an accuracy of 87 % for training and validation and a loss of 0.4 and 0.07 for training and validation using cross-validation with 10 segmentations and 8 epochs per segmentations. It is retrained with the UT dataset raw binarized and relabelling the first and last 20 frames as non-violent. The first with an accuracy for training and validation of 58 % and the second with 78 % and 75 % of accuracy, respectively. Therefore, retraining is not successful with this model.

Training the full IXMAS dataset with the stacked LSTM using cross-validation of 10 divisions and 16 epochs and then training the full UT, achieved an accuracy of 81 % for training and 82 % for validation. Therefore, the suggestion made in (16) led to one of the best results obtained. Applying pretraining between IXMAS and UT using binarization generates improvements on the stacked LSTM model.

### 12.1.6 The Best Approximations

In this section are shown what are considered the two best options obtained for the stacked LSTM and the ConvLSTM models. In the Table 33 is presented the two best models, for them were the ones showing a better generalization with a custom dataset never seen before.

Table 33: Results during training of the models with the best outcome with an independent test, own elaboration

Model	Retrained	Binary	Dataset	Training accuracy [%]	Validation accuracy [%]
<b>LSTM</b>	Yes	Yes	Full UT pretrained with full IXMAS	82	81
<b>ConvLSTM</b>	No	Yes	UT section	99	97

### The Best LSTM

To find the best Stacked LSTM model, it was tested with different conditions such as different deepness or different learning rate, depending on the graphic outputs interpretation. The considered best result was obtained with four layers of deepness, using the Adamax optimizer and a learning rate of  $1e^{-3}$  using the section IXMAS dataset.

After a training of 50 epochs it was interpreted that the learning did not converge, the graphic showed that the model was still learning but the loss did not improve for 3 epochs, as the early stop had a patience of 3, it was decided to augment that patience to 5 and continue. It learned until 140 epochs but still seemed not to converge achieving around 75 % accuracy and 0.8 of loss, so the patience was increased, and the result was that the training could improve.

As seen in Figure 70 and Figure 71 the model has been capable to achieve high results for the labelled with actions datasets. for binarized has been also accurate but with more difficulties. The retrained and binarized model described at the end of 12.1.5 is one of the options most capable to generalize for actions never seen as it will be explained in the chapter 12.2.3.

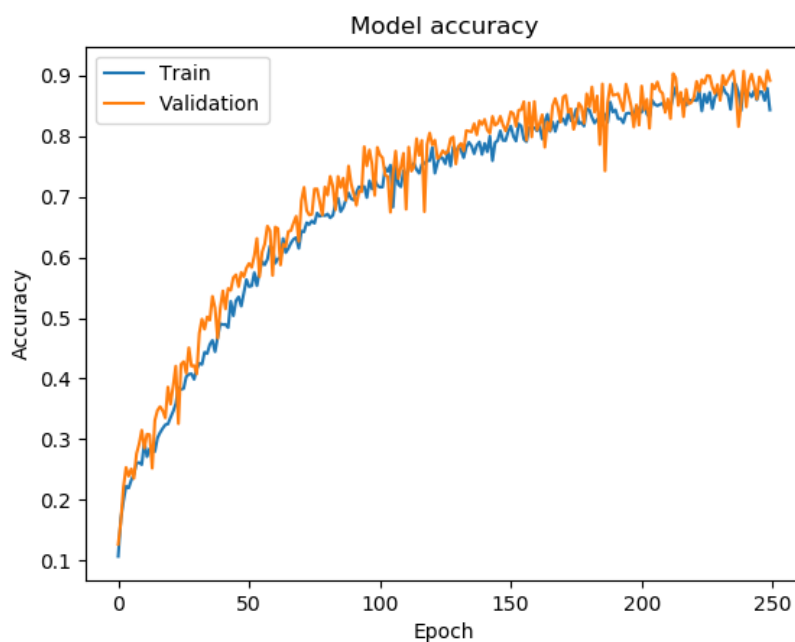


Figure 70: Stacked LSTM model accuracy for 10 time steps and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

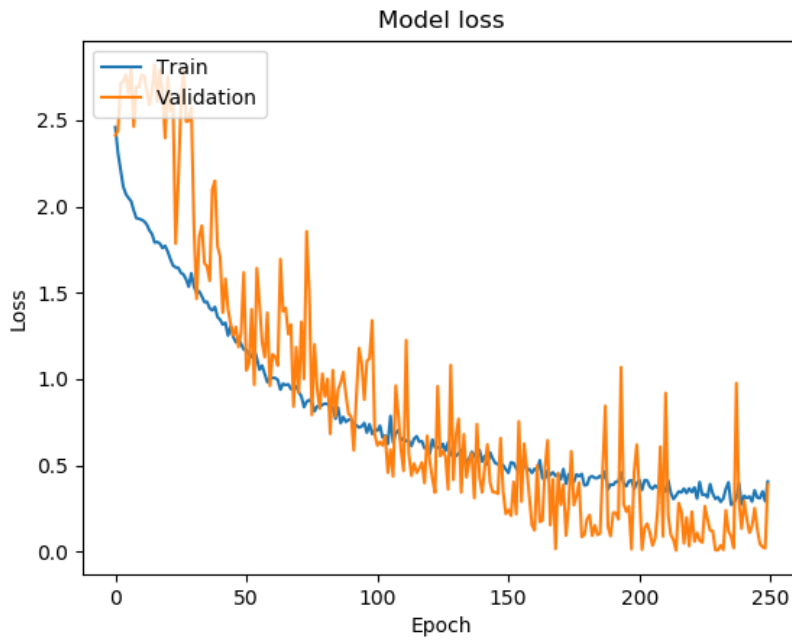


Figure 71: Stacked LSTM model loss for 10 time steps and a learning rate of  $1e^{-3}$ , created using matplotlib during the tests made in this project

**The Best ConvLSTM**

The ConvLSTM model has shown to fit for solving the problem specially with the section IXMAS dataset. The ConvLSTM model contains a ConvLSTM layer, a dropout and a LSTM layer and is interesting how fast it learns compared with the LSTM model that contains four LSTM layers.

The convolutional properties benefit the learning process in training with video dataset and it can be proved with the graphs shown below where it is seen how the model has learnt fast how to solve the problem. The two graphs Figure 72, Figure 73, show the results using a dropout of 0.5, which is a common value. There are some noise movements along the graphs and it learns very fast. It means that the learning was not the best for generalizing and with the speed it was learning it could lead easily to overfitting.

The dropout was changed to 0.75 and in Figure 74 and Figure 75 can be seen how this noise factor was highly reduced. Still this is a very powerful network that learns very fast from big datasets with time dependency.

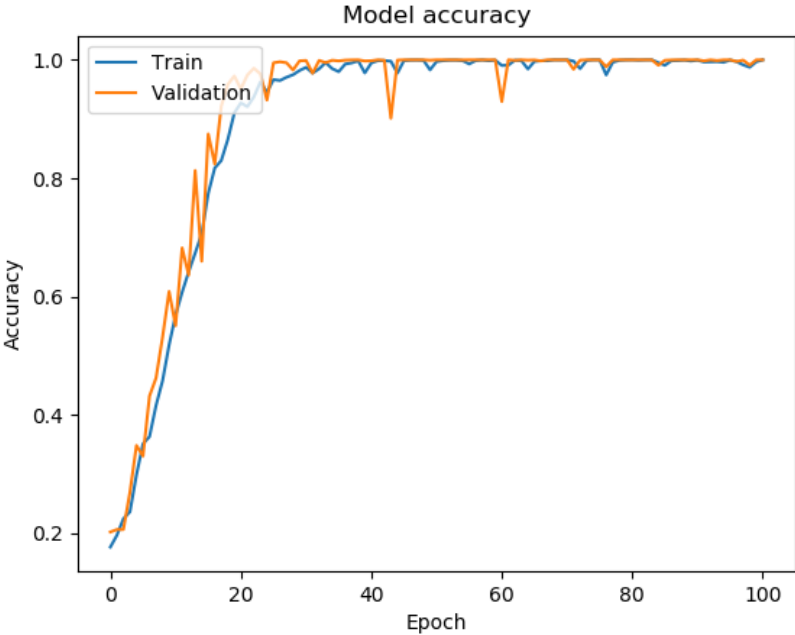


Figure 72: ConvLSTM model accuracy with 0.5 dropout, created using matplotlib during the tests made in this project

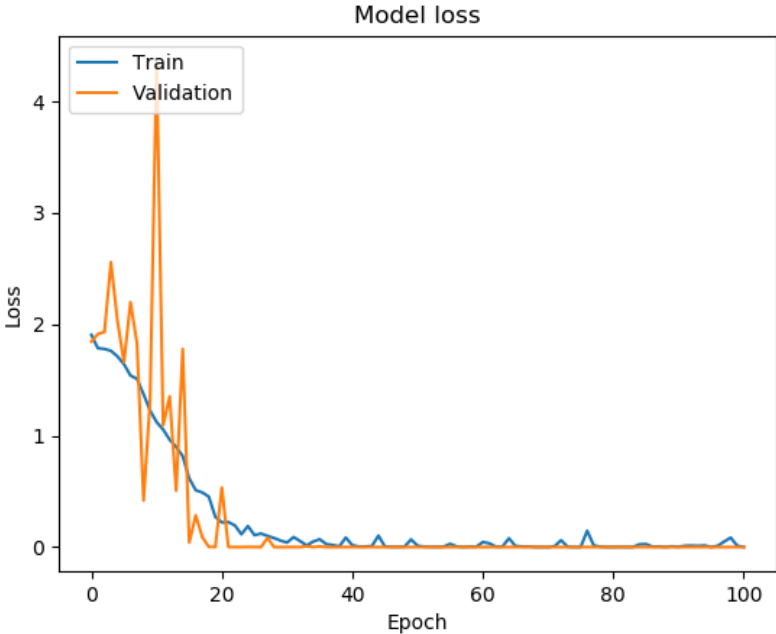


Figure 73: ConvLSTM model loss with 0.5 dropout, created using matplotlib during the tests made in this project

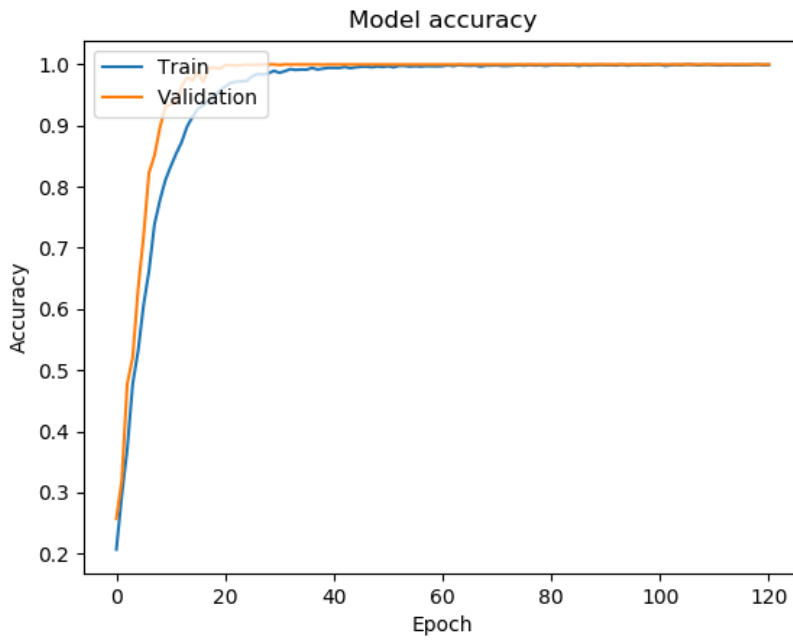


Figure 74: ConvLSTM model accuracy with 0.75 dropout, created using matplotlib during the tests made in this project

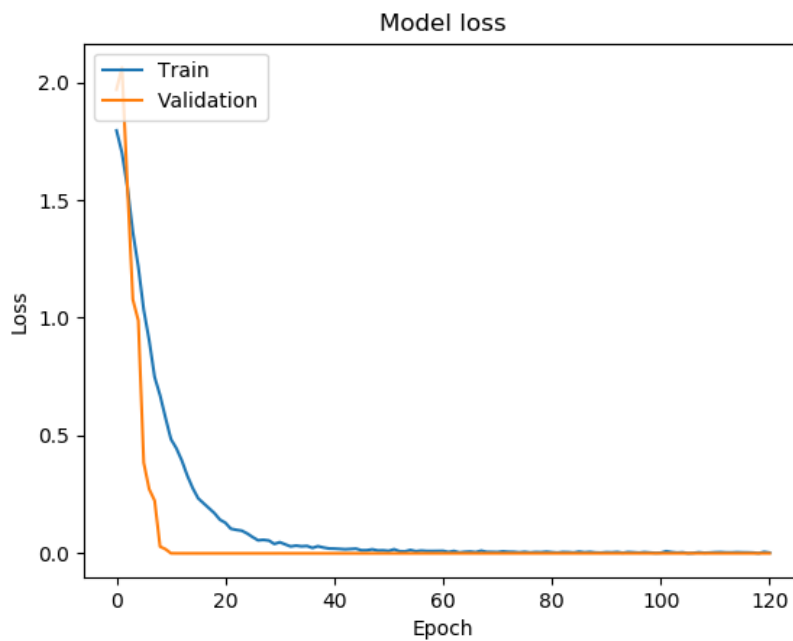


Figure 75: ConvLSTM model loss with 0.75 dropout, created using matplotlib during the tests made in this project

This model has been capable to achieve 99 % and 97 % of accuracy on training and validation with the UT binarized dataset, a difficult combination as models were learning better in general from IXMAS. A big advantage to use ConvLSTM model over the stacked LSTM is the time required for training, this model is more potent for learning thanks to the Convolutional capability explained at 9.2.2. this model is capable to abstract violence on before not seen actions as explained at 12.2.3



Table 35: Noise saved regarding violence in the frames for the LSTM model, own elaboration

LSTM	Violent Frames stored	Noise
<b>1</b>	504	High
<b>8</b>	501	Low
$\frac{1}{2}$ (batch size) = $\frac{1}{2}$ <b>20</b> = <b>10</b>	444	None
$\frac{1}{2}$ (batch size) = $\frac{1}{2}$ <b>30</b> = <b>15</b>	387	None
$\frac{1}{2}$ (batch size) = $\frac{1}{2}$ <b>40</b> = <b>20</b>	309	None

Table 36: Noise saved regarding violence in the frames for the ConvLSTM model, own elaboration

ConvLSTM	Violent Frames stored	Noise
<b>1</b>	331	Low
<b>8</b>	249	Low
$\frac{1}{2}$ (batch size) = $\frac{1}{2}$ <b>20</b> = <b>10</b>	201	None
$\frac{1}{2}$ (batch size) = $\frac{1}{2}$ <b>30</b> = <b>15</b>	201	None
$\frac{1}{2}$ (batch size) = $\frac{1}{2}$ <b>40</b> = <b>20</b>	169	None

### 12.2.2 Timing

The time required for the alarm system to give an output can be considered in real time, as typical surveillance cameras have a frame rate of 30 frames per second and the system is capable to analyse 20 frames in 5 milliseconds (ms) using the ConvLSTM model and 6 milliseconds using the LSTM. The speed depends on the frame rate, for first 20 frames will be accumulated. To detect a violent scene, it is necessary that 10 consecutive frames show violence. In the worst case, this violence appears only at the beginning of the bunch, and 10 more frames will have to be taken to continue with the analysis, then the group will be processed in milliseconds as shown in Table 37.

Table 37: System speed to detect violence in a bunch of 20 frames, own elaboration

Model	Frames	Time [ms]
<b>ConvLSTM</b>	20	5
<b>LSTM</b>	20	6

### 12.2.3 Accuracy Evaluation with a Custom Dataset

To find out the accuracy of the model when confronting the unknown, the trained models are loaded in the alarm system and tested with a long interactions video and with a custom created dataset, which includes scenes and actions that the system has not seen before. In this section the result for the two best models is shown.

Table 38 presents the distribution of action elements on the custom dataset test, their labels and if they are easy to interpret by a human, after asking a group of four volunteers. The action number 1 was grabbing a leg of another person and holding it while this person jumped up and down, moving around with the other leg. It is difficult to determine if this action is violence or not, but the majority

of the requested said that it is playing, therefore not violence. A similar inaccuracy resulted from a video showing action number 5, a hard-fast caress that could also look like a slow fake slap. Here the opinion of the volunteers is balanced with two votes for violence and two votes against.

Table 38: Testing used in the Alarm System, own elaboration

Code	Action	Label	Human accuracy [%]
1	Grabbing leg	Non-violence	75
2	Hug	Non-violence	100
3	Pointing	Non-violence	100
4	pushing	Violence	100
5	Hard-fast caress/ fake slap	Undecided	50

Table 39: Results of the test done on the alarm system for the best models created, own elaboration

Model	1	2	3	4	5	Total accuracy [%]
<b>LSTM</b>	Violence	Non-violence	Non-violence	Violence	Violence	75
<b>ConvLSTM</b>	Non-violence	Non-violence	Non-violence	Violence	Non-violence	100

For the accuracy evaluation with the custom dataset, the action with code 5 in Table 38 is not considered because it is undetermined by human labelling. Table 39 shows the result of the accuracy evaluation. As it happened with the human rating, the action number 1 got differently labelled by the two tested NNs. The LSTM failed in the action where humans also were doubting, so the result of the ConvLSTM is considered better with 100 % accuracy for the tested actions. Therefore, the ConvLSTM model is implemented in the alarm system for further use.

For the evaluation with the long interaction video, it was discovered that the LSTM model tends to consider the close contact actions as violent, especially those new and never trained, such as kissing. Therefore, the ConvLSTM model is the one most suiting the problem.

## 13 Review and Conclusion

This chapter evaluates whether the project's goal and objectives are fulfilled. Therefore, first the completion of the tasks is checked according to the task plan of chapter 8.2. Furthermore, all the important facts and the project's benefit are presented in the project review before the conclusion.



### 13.1 Tasks Review

The tasks of this project are evaluated to determine the quantity of work done. The calculation is based on the quantity of tasks finalized and their priority value calculated in 8.2 for this is an indicator of the objectives' achievement.

Table 40 presents the general results for each task group. The total points are the sum of the priority values of each task on the main task groups, the achieved ones are the sum of the points of those completed. Finally, shown is the percentage (%) of the number tasks accomplished. The result is that four out of six of the main task groups are completed by 100 %. The system preparation and the alarm system lack 30 and 20 points respectively. These missing points are related with two tasks of low priority: The first one is "Apply a license to the project" which is not done yet as the system is not about to be published. The second is "Frame the people on the scenes", but chapter 10.6 classifies this task non-beneficial for the project and therefore the implementation was dropped. As Table 40 is a summary showing the main tasks only, assessment tables showing the sub-tasks can be found in the annex 16.2.

Table 40: General assessment of the project tasks, own elaboration

Code	Main tasks	Total points	Achieved points	Accuracy [%]
1	Organization	745	745	100
2	System preparation	450	420	93
3	Datasets	340	340	100
4	Training comparative studies	825	825	100
5	Alarm system	435	415	95
6	Project documentation	960	960	100
	<b>Sum</b>	<b>3755</b>	<b>3705</b>	<b>98</b>

### 13.2 Project Review

The project **organization** and working based on the PDCA method resulted being useful to improve the system development as after each experiment the result was analysed. Dependent on the outcome, the models were constantly updated leading to the best experimental result achievable, based on the state of the art and the requirements presumed for this application. The project has finished with two strong models that are showing a good accuracy with the validation datasets and the custom dataset, which verifies the functionality of the software.

The **system preparation**, especially the state of the art research, occupied a long period of the project duration. Nevertheless, the invested time was paid back because the gained expertise allowed a good choice of tools and accelerated the following system setup and development significantly.

Using a greyscale **dataset** appeared beneficial because it does not reduce the accuracy, due to the colour not being determinant, and lowers computational efforts and memory space used to process

the images. Optimizing the size of the datasets to achieve a good result with short trainings was very useful while the comparative testing. Another beneficial modification of the UT dataset is the non-violence-labelling of the first and last frames of all videos where the violence action is still not happening per more than 20 frames. Generally, it is recommended to use an equilibrated distribution of each label in a training dataset. The applied relabelling changed this balance of violence and non-violence frames and lead to an improved training result. This experiment proves that it is more beneficial to reduce wrong labelled frames, for example the time people are standing before starting a violent interaction, than keeping an exact equilibrium. Also, the binarization of the datasets to violence/non-violence labels lead to better test results, than keeping an individual label for each kind of action. The model was also learning faster at the beginning of each training. This is interpreted as having less labels makes it easier to find features to correctly assign the label. The dataset IXMAS was generally related with better accuracy results than the UT dataset due to the simplicity, the smaller size per frame, and being a bigger dataset.

One of the biggest improvements while **training comparative studies** resulted from applying data normalization. This breakthrough improved the accuracy of the model training with the IXMAS dataset from a 30 % to 85 %. Furthermore, the cross-validation positively influenced the accuracy result over the epochs. Using this method, the accuracies where incremented in a more similar way than with other methods tested. The distribution of validation and training was more even because the whole dataset was used for training and validation. The categorical crossentropy, compared to MSE loss, was fastening the learning with the big video datasets used for training. The code for this project is created based on the attributes of scalability and reusability to ease adding new controllers and modifiers to the current datasets and new ones. It enables adding new models and trainer code and contains an extendable data generator providing data in customized structures. Following experiments showed that the optimized batch size lies between 40 and 80, smaller sizes gave worse results. To reach the best memorization for the LSTM layers, time steps of 10 are proposed. When using a higher time step value, the validation loss tends to progress worse. Finally comparing the two best models: The ConvLSTM network achieved the same accuracy result as the LSTM, but within 20 training epochs instead of 200 epochs. The investigation made to select the best settings for the NNs and the data restructuration lead to the possibility to achieve a good result with an independent dataset using small LSTM and ConvLSTM models.

The developed **alarm system** can be used with a surveillance camera. To prove its effectivity, the system was evaluated with a custom recorded video dataset that was never used while the training and validation steps. When analysing unknown actions, as for example kissing, both NNs made a few mistakes but were in general able to detect violence. An important setting to avoid mistakes is that violence must be detected for at least 10 consecutive frames using a batch size of 20 before the alarm is triggered. This avoids mistakes as, for example, a single frame giving a violence output in a

full video of non-violent actions. Dependent on the model chosen to assess the surveillance camera's videos, also the sensibility to violence may differ. To summarize this part of the conclusion: The alarm system is working but needs some fine-tuning regarding the detection sensibility.

Finally, the **project documentation** contains all the important results of this study. It preserves the gained knowledge and creates a good base for further development or commercialization of this system.

### 13.3 Conclusion

To fulfil the first objective, "Detecting danger situations in video in an automatized way" two models were developed that are capable to detect violence in an independent dataset not seen before. These models are implemented in a software that systematically reads video data and returns the result of the training for batches of 20 frames in approximately 6 milliseconds. The second and third objectives, "Saving the information related with the possible aggression" and "Generating an alarm to warn the competent authority" were accomplished in the alarm system. After detecting a violence situation, the software generates a sound alarm to inform the personnel in charge and saves related frames of the situation, including name of the camera, date and time, for the use as evidence.

For all objectives achieved, it is assumed that the system is capable to fulfil the project goal "to improve the possible detection time of violence by bringing a tool that can help to stop bullying situations, in a fast way". Currently it is impossible, of course, to verify the accomplishment of the goal, because a larger field test or the software release followed by a student are necessary to gain reliable data about bullying statistics and reaction times.

## 14 Future Work

In this chapter, ideas for further improvement and recommendations on how to proceed are suggested:

A license for the developed software is considered before public release but before there are further optimization steps to be processed.

Applying background subtraction combined with NN is very interesting as it recognizes the presence of the person and it can bring the opportunity to focus on the analysis of the interaction ignoring the background. This technique could be approached using the OpenCV library. A good reference for this task can be (60) and (61), who already introduced studies regarding background subtraction.

To increase the accuracy of the model regarding group violence, related videos should be added to the training datasets. To reduce the required training time of those large datasets, decreasing the frames per second of the videos may be considered. The alarm system is capable to process batches of 40 frames in less than one second and therefore the system is capable to work with typical

surveillance cameras videos in real time. This should be subject of further tests with real time camera input.

The videos taken from surveillance cameras can also be an interesting input for training, because it is composed by real situations. Those actions are more natural than typical interpretations used for training and therefore the effectivity of the model may be increased. These videos could be a source applied via SL learning style. This means labelling them and applying them to optimize the accuracy of the already trained models. To proceed on this path, the allowance of an institution where the surveillance takes places and of the people appearing is needed to respect the regulation (EU) 2016/679, commonly known as General Data Protection Regulation (GDPR) (62).

It is a time-consuming task to label a large amount of surveillance camera videos. If the software is already working stable with good accuracy, the RL learning style can be applied after the release. This method requires big amount of data, which is provided by the quantity of surveillance cameras installed, but requires personnel give feedback to the system to let the AI learn from it.

## 15 References

1. UNESCO. Sustainable Development Goal 4 : 4.a.2 Percentage of students experiencing bullying in the last 12 months. [Online] [Cited: 25 03 2020.] <http://data.uis.unesco.org/index.aspx?queryid=3624>.
2. Spain, Amnesty International. España: acoso escolar, un problema invisible que precisa un sistema de denuncias útil de verdad. [Online] 5 June 2019. [Cited: 25 03 2020.] <https://www.es.amnesty.org/en-que-estamos/noticias/noticia/articulo/espana-acoso-escolar-un-problema-invisible-que-precisa-un-sistema-de-denuncias-util-de-verdad/>.
3. Assembly, United Nations General. *Universal Declaration of Human Rights*. Paris : s.n., 10 December 1948. Declaration.
4. 2, ISO/TC 176/SC. Quality management systems - Requirements DIN EN ISO 9001:2015-11 . November 2015. Vol. 5.
5. *Violence detection using Oriented Violent Flows*. Gao, Yuan, et al. 24 February 2016, Image and Vision Computing, pp. 37-41.
6. *Human violence recognition and detection in surveillance video*. Bilinski, P and Bremond, F. Colorado Springs : IEEE, 2016. 13th International Conference of Advanced Video and Signal Based Surveillance. pp. 30-36.
7. *Violence Detection in Surveillance Video-A Survey*. Naik, Anuja Jana and Gopalakrishna, M. T. 2016, International Journal of Latest Research in Engineering and Technology, pp. 11-17. 2454-5031.
8. *Violence Detection in Video Using Computer Vision Techniques*. Bermejo, Enrique, et al. 2011. International Conference on Computer Analysis of Images and Patterns. pp. 332-339.
9. *A Sensor approach for Violence Detection in Smart Cities Using Deep Learning*. Baba, Marius, et al. Basel : s.n., 8 April 2019, Sensors, Vol. 19(7).
10. van Veen, Fjodor and Leijnen, Stefan. The Neural Network Zoo. *The Asimov Institute*. [Online] 14 September 2016. [Cited: 16 May 2020.] <https://www.asimovinstitute.org/neural-network-zoo/>.
11. *Understanding the Principles of Recursive Neural Networks: A Generative Approach to Tackle Model Complexity*. China, Alejandro. s.l. : Springer, Berlin, Heidelberg, 2009. ICANN 2009: Artificial Neural Networks. Vol. 5768, pp. 952-963.
12. Understanding LSTM Networks. *Colah's blog*. [Online] 2015. [Cited: 21 April 2020.] <https://colah.github.io/posts/2015-08-Understanding-LSTMs>.

13. Wang, Chi-Feng. towards data science. *The Vanishing Gradient Problem*. [Online] [Cited: 13 April 2020.] <https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484>.
14. *Gradient-based Learning Applied to Document Recognition*. Lecun, Y., et al. 1998. Proceedings of the IEEE. Vols. 86, no. 11, pp. 2278-2324.
15. *ImageNet Classification with Deep Convolutional Neural Networks*. Krizhevsky, Alex, Sutskever, Ilya and Hinton, Geoffrey E. s.l. : Neural Information Processing Systems, 2012.
16. Dai, Qi, et al. Fudan\_Huawei at MediaEval 2015: Detecting Violent Scenes and Affective Impact in Movies with Deep Learning. s.l. : MediaEval, 2015.
17. Howard, Andrew G., et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. arXiv, 1704.04861.
18. Iandola, Forrest N., et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. 2016. arXiv, 1602.07360.
19. *Long Short-Term Memory*. Hochreiter, Sepp and Schmidhuber, Jürgen. November 1997, Neural Computation, Vols. 9,n°8.
20. Hyndman, Rob J. and Athanasopoulos, George. *Forecasting: Principles and Practice*. Australia : s.n., 2018.
21. Chollet, François and and others. Keras. *Keras*. [Online] 2015. [Cited: 23 April 2020.] <https://keras.io>.
22. Hinton, Geoffrey, Srivastava, Nitish and Swersky, Kevin. Neural Networks for Machine Learning. Lecture 6a. Overview of mini-batch gradient descent.
23. *Adam: A Method for Stochastic Optimization*. Kingma, Diederik P. and Ba, Jimmy. San Diego : s.n., 2015. 3rd International Conference for Learning Representations. arXiv, 1412.6980.
24. Usage of optimizers. [Online] [Cited: 01 April 2020.] <https://keras.io/optimizers/>.
25. *On the Importance of Initialization and Momentum in Deep Learning*. Sutskever, Ilya, et al. 2013. Proceedings of the 30th International Conference on Machine Learning. Vol. 28, pp. 1139-1147.
26. Hinton, Geoffrey, Srivastava, Nitish and Swersky, Kevin. Neural Networks for Machine Learning. [ed.] University of Toronto Computer Science. *Lecture 6a Overview of mini-batch gradient descent*.
27. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. Clevert, Djork-Arné, Unterthiner, Thomas and Hochreiter, Sepp. 2016. ICLR.

28. *Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units*. Shang, Wenling, et al. 2016.
29. *Convolutional Deep Belief Networks on CIFAR-10*. Krizhevsky, Alex. 8 May 2012.
30. Usage of loss functions. *Keras Documentation*. [Online] [Cited: 04 April 2020.] <https://keras.io/losses/>.
31. Goodfellow, Ian, Bengio, Yoshua and Courville, Aaron. *Deep Learning*. 2016. 978-0262035613.
32. Cherry Servers. [Online] [Cited: 19 May 2020.] <https://www.cherryservers.com/>.
33. Hinton, Geoffrey E., et al. Improving neural networks by preventing co-adaptation of feature detectors. 2012. arXiv preprint arXiv:1207.0580.
34. *A Survey on Vision-Based Human Action Recognition*. Poppe, Ronald Walter. 6, June 2010, Image and Vision Computing, Vol. 28, pp. 976-990.
35. *Fast Violence Detection in Video*. Deniz, Oscar, et al. Lisbon : Institute of Electrical and Electronics Enigneers, 2014. pp. 478-485. 978-9-8975-8133-5.
36. *Two-person Interaction Detection Using Body-Pose Features and Multiple Instance Learning*. Yun, Kiwon, et al. Rhode Island : s.n., 2012. Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference.
37. *Free Viewpoint Action Recognition using Motion History Volumes*. Weinland, Daniel, Ronfard, Remi and Boyer, Edmond. s.l. : Elsevier, 13 July 2006, Computer Vision and Image Understanding, Vols. 104 (2-3), pp. 249-257.
38. Ryoo, M. S. and Aggarwal, J. K. UT-Interaction-Dataset. *ICRP contest on Semantic Description of Human Activities*. 2010.
39. *Violent Flows: Real-Time Detection of Violent Crowd Behavior*. Hassner, Tal, Itcher, Yossi and Kliper-Gross, Orit. Rhode Island : IEEE, The Institute of Electrical and Electronics Engineers, 2012. Conference on Computer Vision and Pattern Recognition .
40. The Movies Dataset. [Online] [Cited: 5 May 2020.] <https://www.kaggle.com/rounakbanik/the-movies-dataset>.
41. *Spatio-Temporal Relationship Match: Video Structure Comparison for Recognition of Complex Human Activities*. Ryoo, M. S. and Aggarwal, J. K. Kyoto : The Institute of Electrical and Electronics Engineers, 2009. IEEE International Conference on Computer Vision.

42. *Violence Detection in Movies*. Chen, Liang-Hua, et al. Singapore : The Institute of Electrical and Electronics Engineers, 2011. Eighth International Conference Computer Graphics, Imaging and Visualization. pp. 119-124.
43. Clarin, Christine T., et al. DOVE : Detection of Movie Violence using Motion Intensity Analysis on Skin and Blood. 2016.
44. *Audio-visual Content-based Violent Scene Characterization*. Nam, Jeho, Alghoniemy, Masoud and Tewfik, Ahmed H. 1998, Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269), Vol. 1, pp. 353-357.
45. *CASSANDRA: Audio-Video Sensor Fusion for Aggression Detection*. Zajdel, W., Krijnders, J. D. and Gavrilu, D. M. London : The Institute of Electrical and Electronics Engineers, 2007. Conference on Advanced Video and Signal Based Surveillance. pp. 200-205.
46. *Weakly-Supervised Violence Detection in Movies with Audio and Video Based Co-training*. Lin, Jian and Wang, Weiqiang. 2009. Advances in Multimedia Information Processing, Pacific-Rim Conference on Multimedia. pp. 930-935.
47. Herath, Samitha, Harandi, Mehrtash and Porikli, Fatih. Going Deeper into Action Recognition: A Survey. 2017. arXiv, 1605.04988v2.
48. *Speech Recognition with Deep Recurrent Neural Networks*. Graves, Alex, Mohamed, Abdelrahman and Hinton, Geoffrey. s.l. : University of Toronto, 22 March 2013.
49. *Convolutional Learning of Spatio-temporal Features*. Taylor, Graham W., et al. 2010. Computer Vision - ECCV 2010. pp. 140-153.
50. Wang, Limin, et al. Temporal Segment Networks: Towards Good Practices for Deep Action Recognition. 2016. arXiv, 1608.00859.
51. Donahue, Jeff, et al. Long-term Recurrent Convolutional Networks for Visual Recognition and Description. 2014. arXiv, 1411.4389.
52. *What is the best multi-stage architecture for object recognition?* Jarrett, K., et al. s.l. : The Institute of Electrical and Electronics Engineers, 2009. International Conference on Computer Vision.
53. Wang, Chi-Feng. towards data science. *The Vanishing Gradient Problem*. [Online] 2019. [Cited: 12 April 2020.] <https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484>.
54. *Representation Learning on Large and Small Data*. Chou, Chun-Nan, et al. 2017.



55. Mastery, Machine Learning. How to use Learning Curves to Diagnose Machine Learning Model Performance. [Online] [Cited: 29 March 2020.] <https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/>.
56. *CIFAR-10 Classification using Intel® Optimization for TensorFlow*. s.l. : intel, 12 December 2017.
57. Kuhn, Max and Johnson, Kjell. *Applied Predictive Modeling*. 2013. ISBN 978-1-4614-6849-3.
58. *tensorflow transition to gpu version*. [Online] 2019. [Cited: 13 April 2020.] <https://stackoverflow.com/questions/58187677/tensorflow-transition-to-gpu-version>.
59. Jarosciak, Jozef. *How to resolve TensorFlow 2.0 Error – Could not load dynamic library ‘cudart64\_100.dll’*. [Online] 2019. [Cited: 13 April 2020.] [https://www.joe0.com/2019/10/19/how-resolve-tensorflow-2-0-error-could-not-load-dynamic-library-cudart64\\_100-dll-dlerror-cudart64\\_100-dll-not-found/](https://www.joe0.com/2019/10/19/how-resolve-tensorflow-2-0-error-could-not-load-dynamic-library-cudart64_100-dll-dlerror-cudart64_100-dll-not-found/).
60. *Person-on-person Violence Detection in Video Data*. Datta, Ankur, Shah, Mubarak and Lobo, Niels da Vitoria. 2002. Vol. 1, pp. 433-438. 0-7695-1695-X.
61. *A Hibrid Framework Combining Background Substraction and Deep Neural Networks for Rapid Person Detection*. Kim, Chulyeon, et al. 10 July 2018, Journal of Big Data 5, Vol. 22.
62. *REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL*. Union, The European Parliament and the Council of the European. 27 April 2016, Official Journal of the European Union.

# 16 Annex

## 16.1 Gantt Diagram

Caption	
Planned	
Holidays	
In time	
Delayed	

Figure 76: Legend of the color code used in the Gantt Diagram, own elaboration

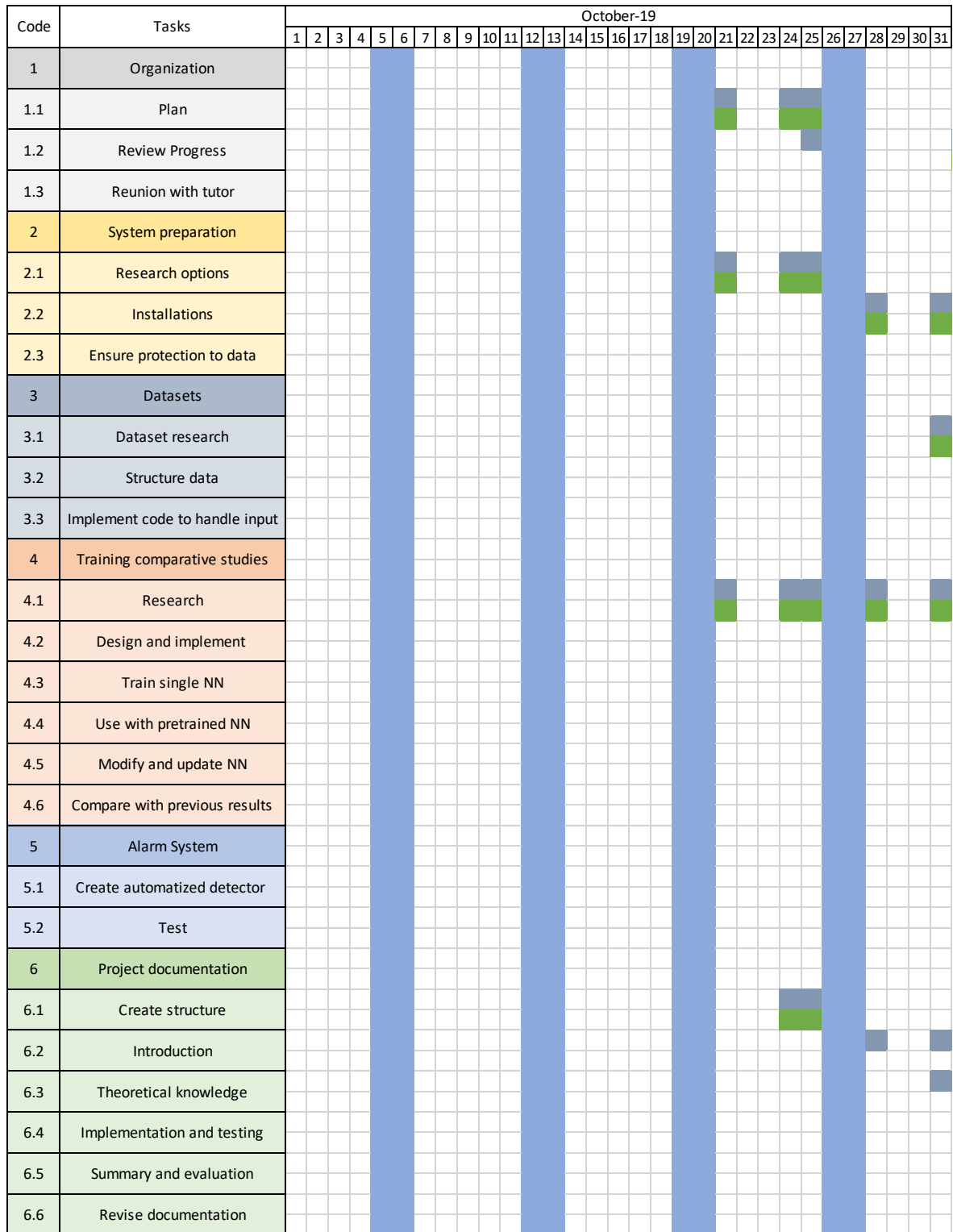


Figure 77: Gantt diagram for the month of October 2019, own elaboration

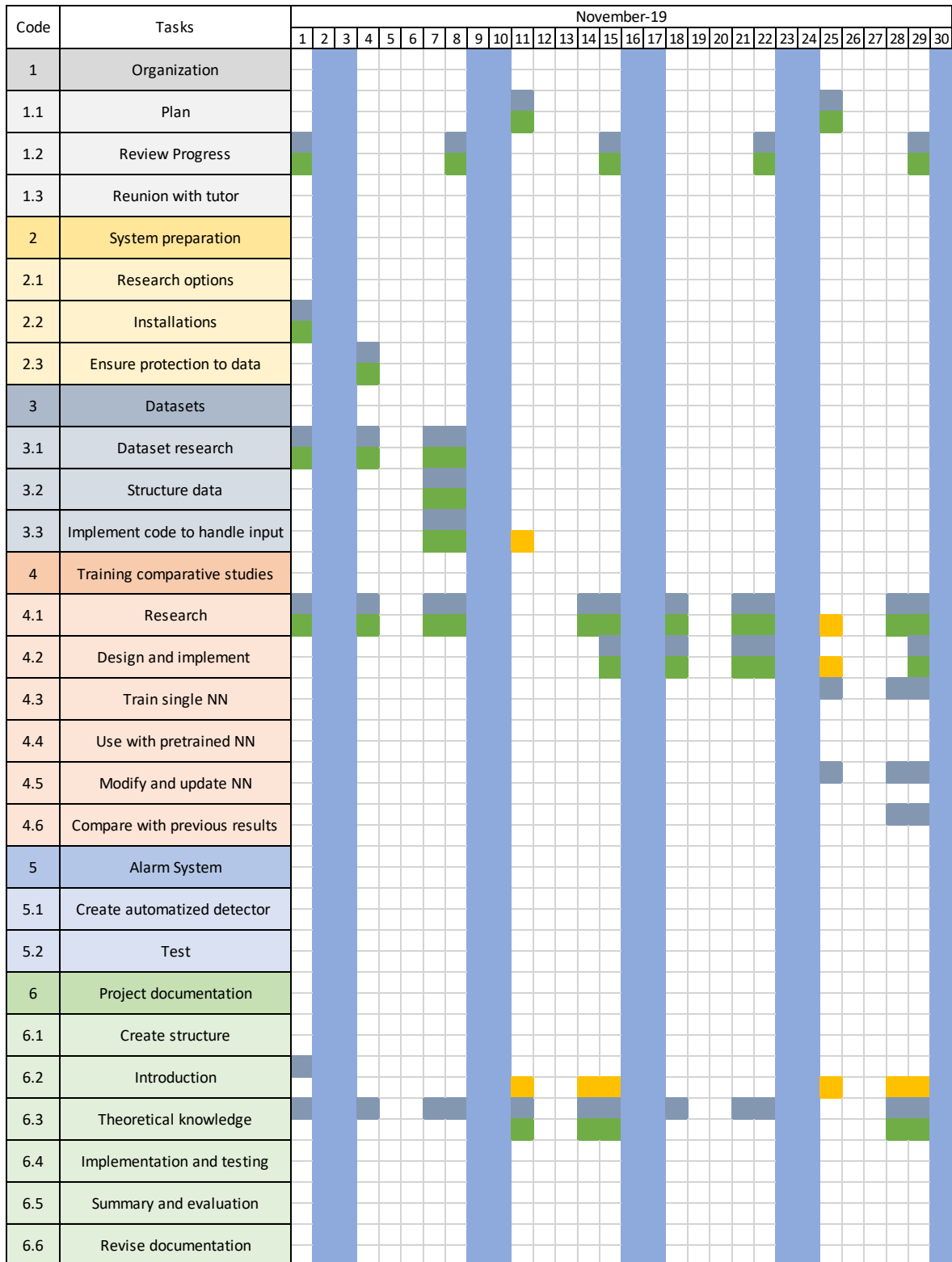


Figure 78: Gantt diagram for the month of November 2019, own elaboration

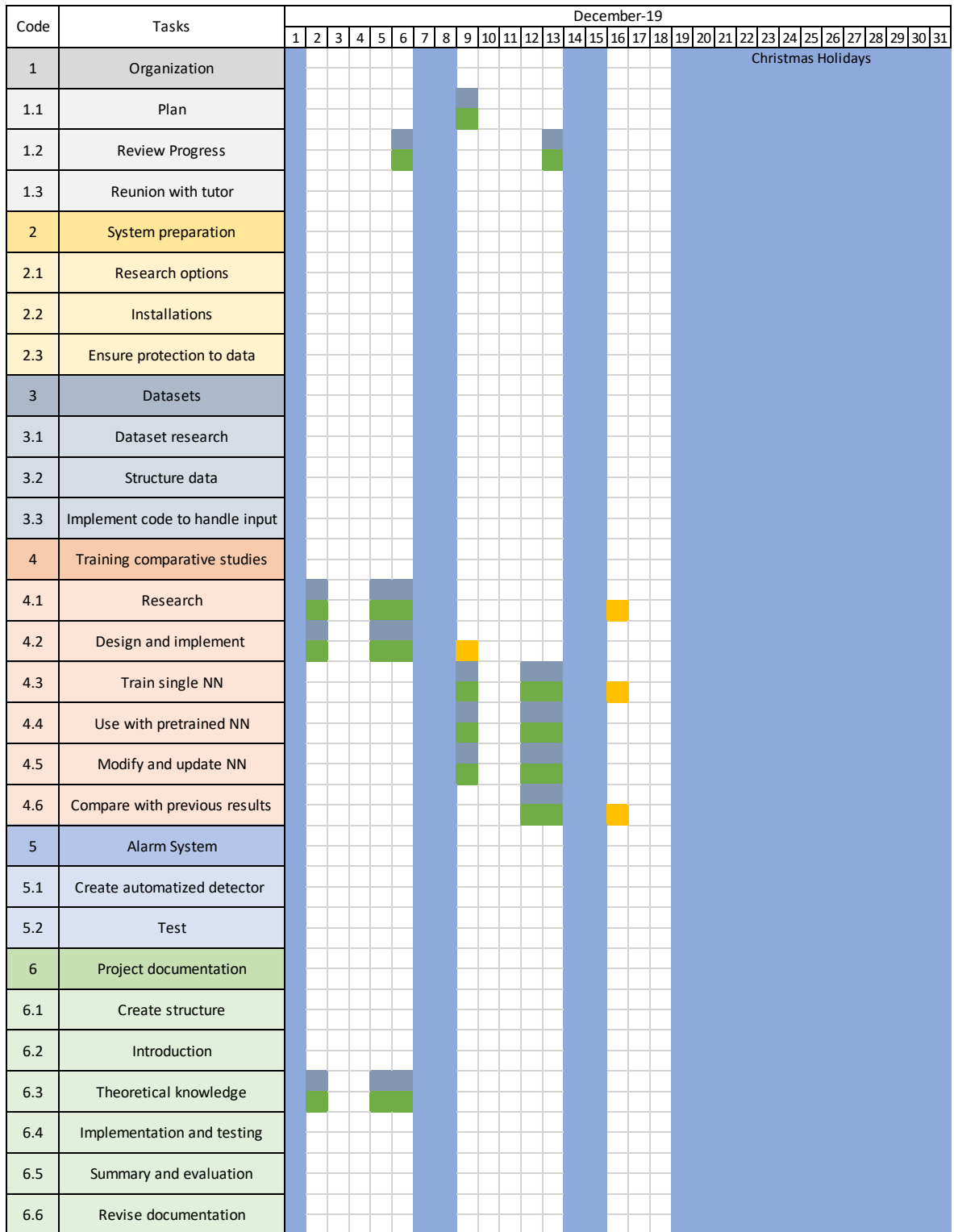


Figure 79: Gantt diagram for the month of December 2019, own elaboration

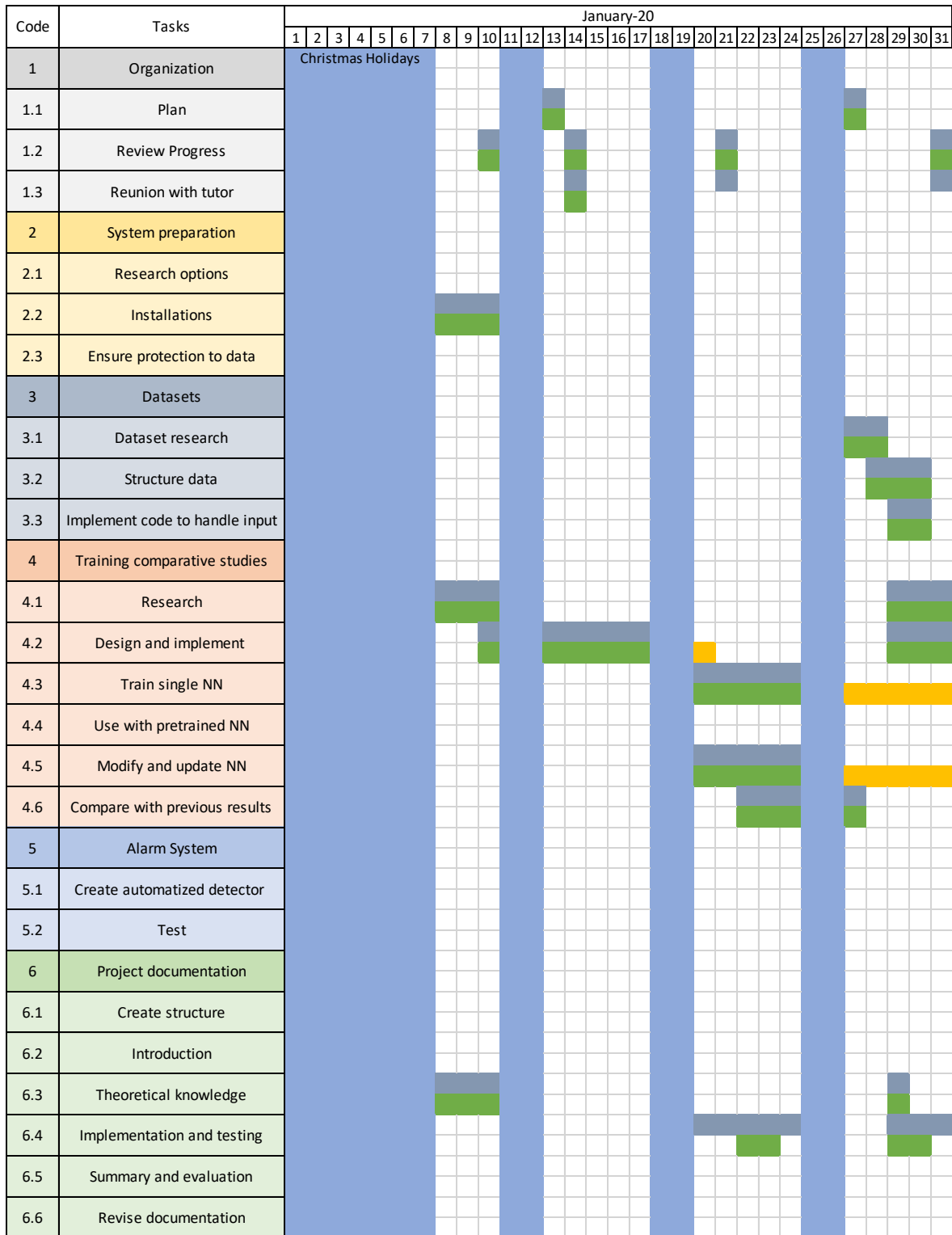


Figure 80: Gantt diagram for the month of January 2020, own elaboration

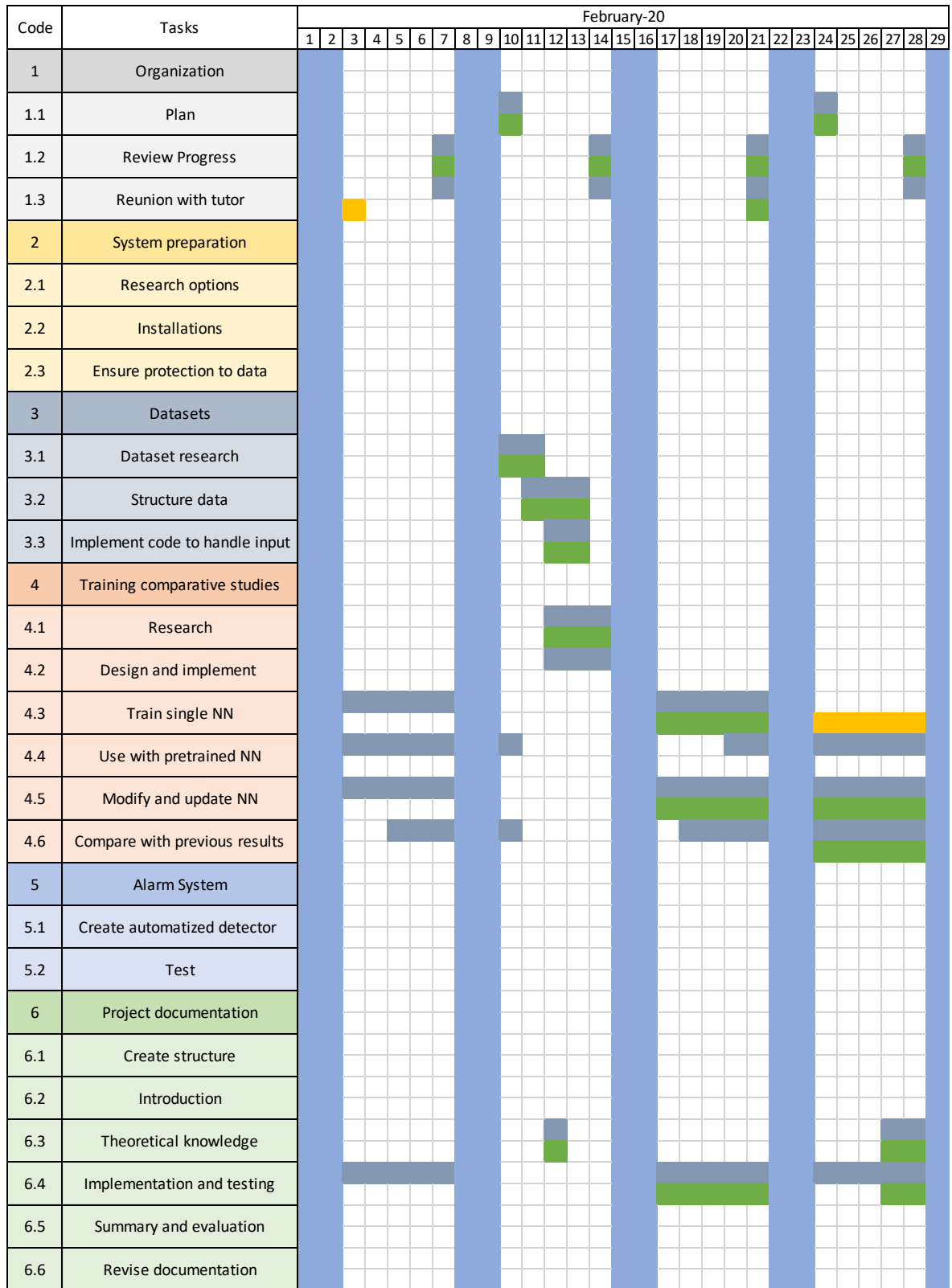


Figure 81: Gantt diagram for the month of February 2020, own elaboration

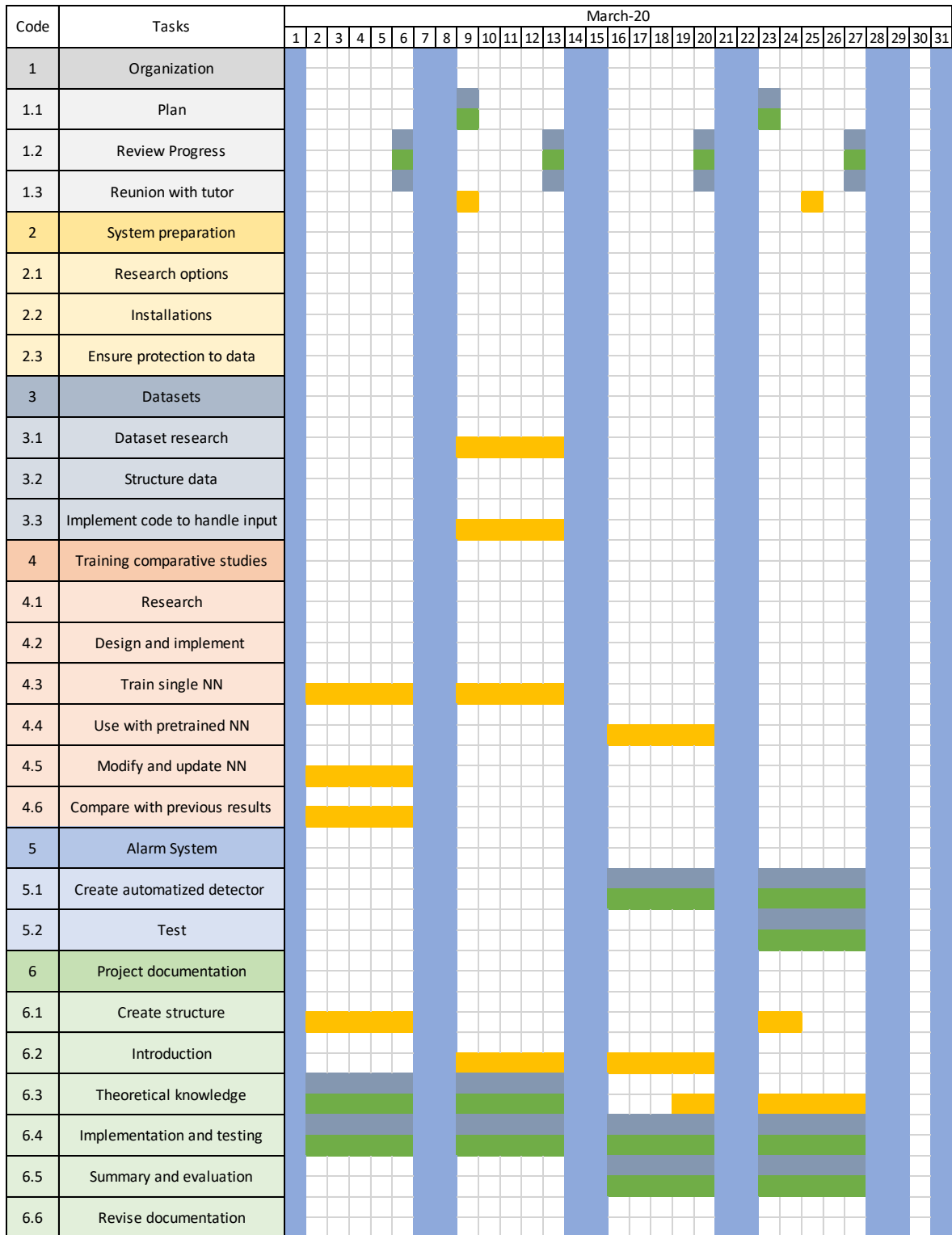


Figure 82: Gantt diagram for the month of March 2020, own elaboration



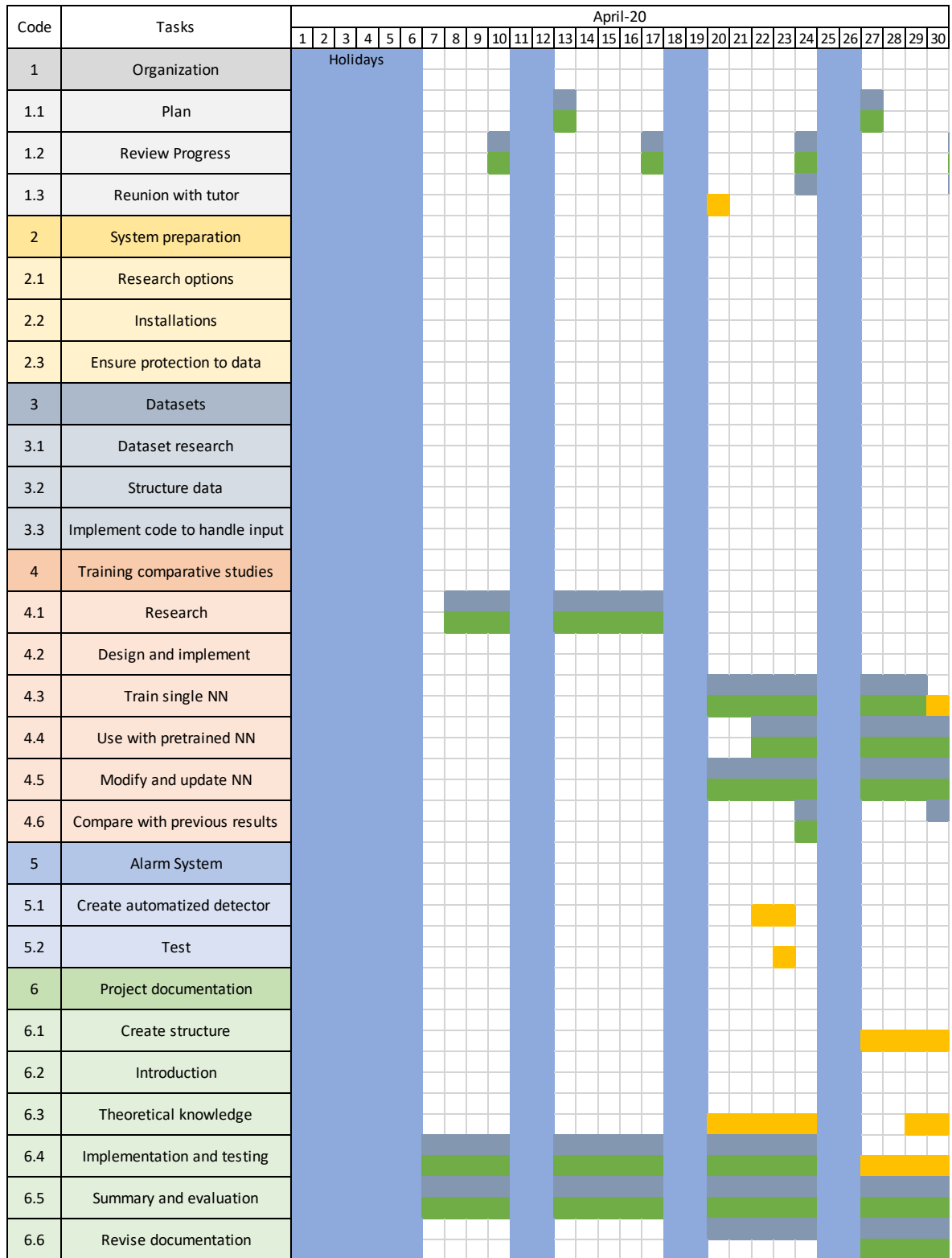


Figure 83: Gantt diagram for the month of April 2020, own elaboration

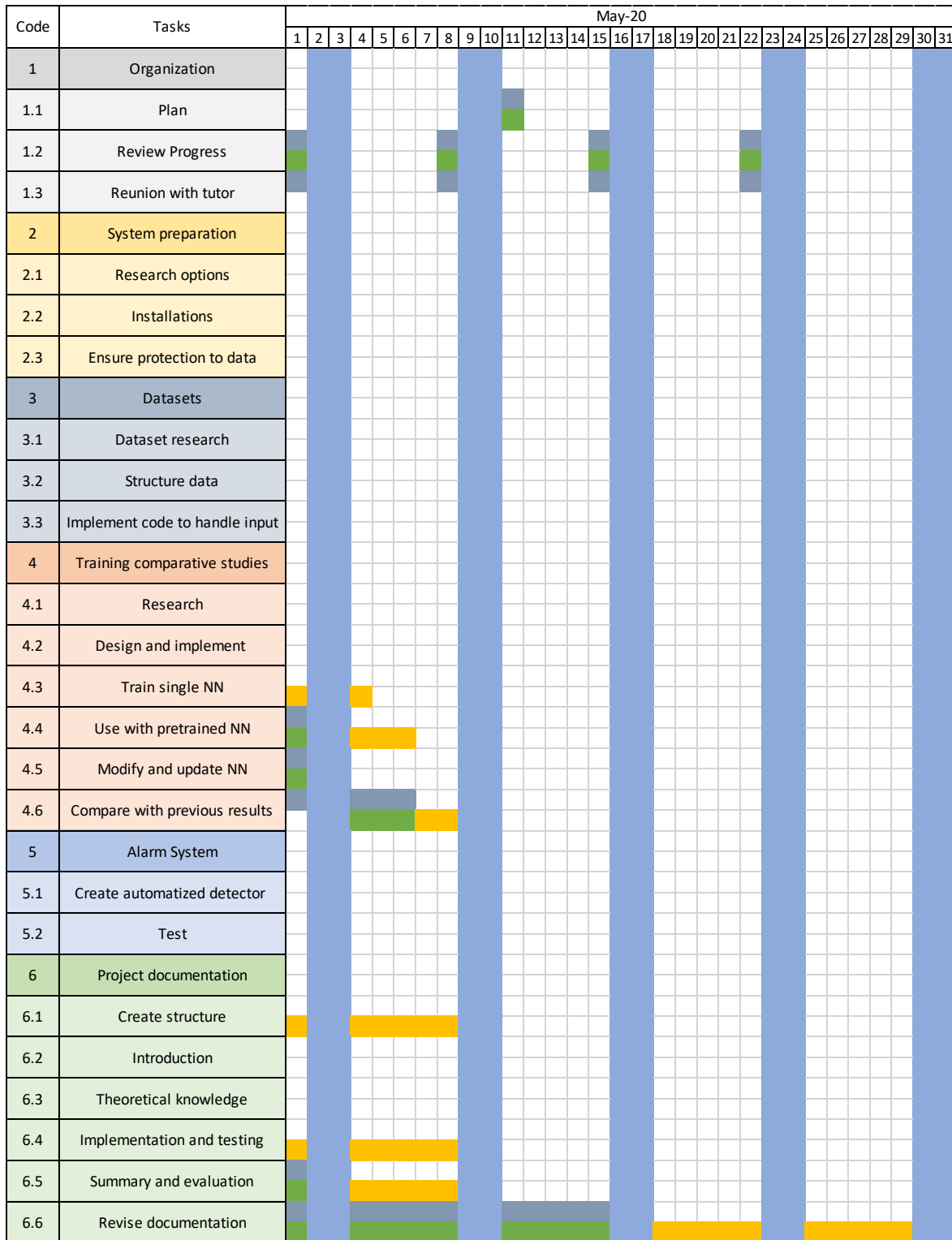


Figure 84: Gantt diagram for the month of May 2020, own elaboration

## 16.2 Detailed Task Review

### 16.2.1 Organization

Table 41: Priority of tasks required for the organization of the project, own elaboration

1. Organization				
Code	Task	Details	PRIO	Done
1.1	Plan	Produce the list of tasks determinant to fulfil the objectives of the project	100	Yes
		Organize the tasks depending on their importance for the project	120	Yes
		Create a time plan	75	Yes
1.2	Review Progress	Summarise the status of the project	90	Yes
		Check if the progress reached fulfils the plan	90	Yes
1.3	Reunion with tutor	Explain progress to the tutor	90	Yes
		Explain important changes to the tutor	90	Yes
		Discuss the next steps with the tutor	90	Yes

### 16.2.2 System Preparation

Table 42: Priority of tasks required for the system preparation of the project, own elaboration

2. System preparation				
Code	Task	Details	PRIO	Done
2.1	Research options	Choose the favourite options for building and running the project	100	Yes
2.2	Installations	Install required software	100	Yes
		Make the required installations for the dependency with libraries and frameworks	100	Yes
2.3	Ensure protection to data	Apply a license to the project	30	No
		Use a control version	120	Yes

### 16.2.3 Datasets

Table 43: Assessment of tasks related with the datasets of the project, own elaboration

3. Datasets				
Code	Task	Details	PRIO	Done
3.1	Dataset research	Acquire datasets that are of use for the project	80	Yes
		Analyse scholar literature (journal articles, conference proceedings, books, ...) related to works where those datasets where used	80	Yes
3.2	Structure data	Design suitable datasets for the model from the original video datasets to use during the training	100	Yes
3.3	Implement code to handle input	Create code to automatically modify big datasets, reorganize and relabel	80	Yes

### 16.2.4 Training Comparative Studies

Table 44: Assessment of tasks required for the training comparative studies of the project, own elaboration

4. Training comparative studies				
Code	Task	Details	PRIO	Done
4.1	Research	Analyse the related theory using literature (journal articles, conference proceedings, books, ...)	60	Yes
		Analyse the current state of the art using literature (journal articles, conference proceedings, books, ...)	60	Yes
		Compare different model approximations	60	Yes
4.2	Design and implement	Apply learned information to design an own NN	80	Yes
		Implement the NN	100	Yes
4.3	Train single NN	Experiment with different NN	45	Yes
		Execute trainings of the NN with different datasets	45	Yes
4.4	Use with pretrained NN	Execute retraining of a model with different datasets	45	Yes
4.5	Modify and update NN	Modify NN to improve the results based on the evaluation	120	Yes
4.6	Compare with previous results	Analyse the results obtained from the trainings	60	Yes
		Compare the accuracy of different datasets	60	Yes
		Analyse how different values affect the results of the training	90	Yes

### 16.2.5 Alarm System

Table 45: Assessment of tasks required for the alarm system of the project, own elaboration

5. Alarm system				
Code	Task	Details	PRIO	Done
5.1	Create automatized detector	Use trained models to evaluate violence detector in video	100	Yes
		Create interface	25	Yes
		Frame the people on the scenes	25	No
		Store the detected violence video sections	90	Yes
		Signalize if a violence situation is detected by the system	120	Yes
5.2	Test	Test the alarm with an independent dataset	75	Yes

### 16.2.6 Project Documentation

Table 46: Assessment of tasks required for the documentation of the project, own elaboration

6. Project documentation				
Code	Task	Details	PRIO	Done
6.1	Create structure	Customize the documentation layout to fulfil the layout of the University of Alicante requirements for final projects	120	Yes
6.2	Introduction	Generate project introduction documentation	80	Yes

<b>6.3</b>	Theoretical knowledge	Generate documentation of the studies obtained related to the project	40	Yes
		Organize the bibliography into a folder	30	Yes
<b>6.4</b>	Implementation and testing	Generating documentation of the programming progress	60	Yes
		Generating documentation of the tests realized	60	Yes
		Generating documentation of the result and interpretation	80	Yes
<b>6.5</b>	Summary and evaluation	Evaluating the results of the training comparative studies	100	Yes
		Evaluating the results of the alarm system	100	Yes
		Assessment of the project objectives	90	Yes
		Conclusion and future work	120	Yes
<b>6.6</b>	Revise documentation	Correct mistakes in the documentation	80	Yes

### 16.3 UML

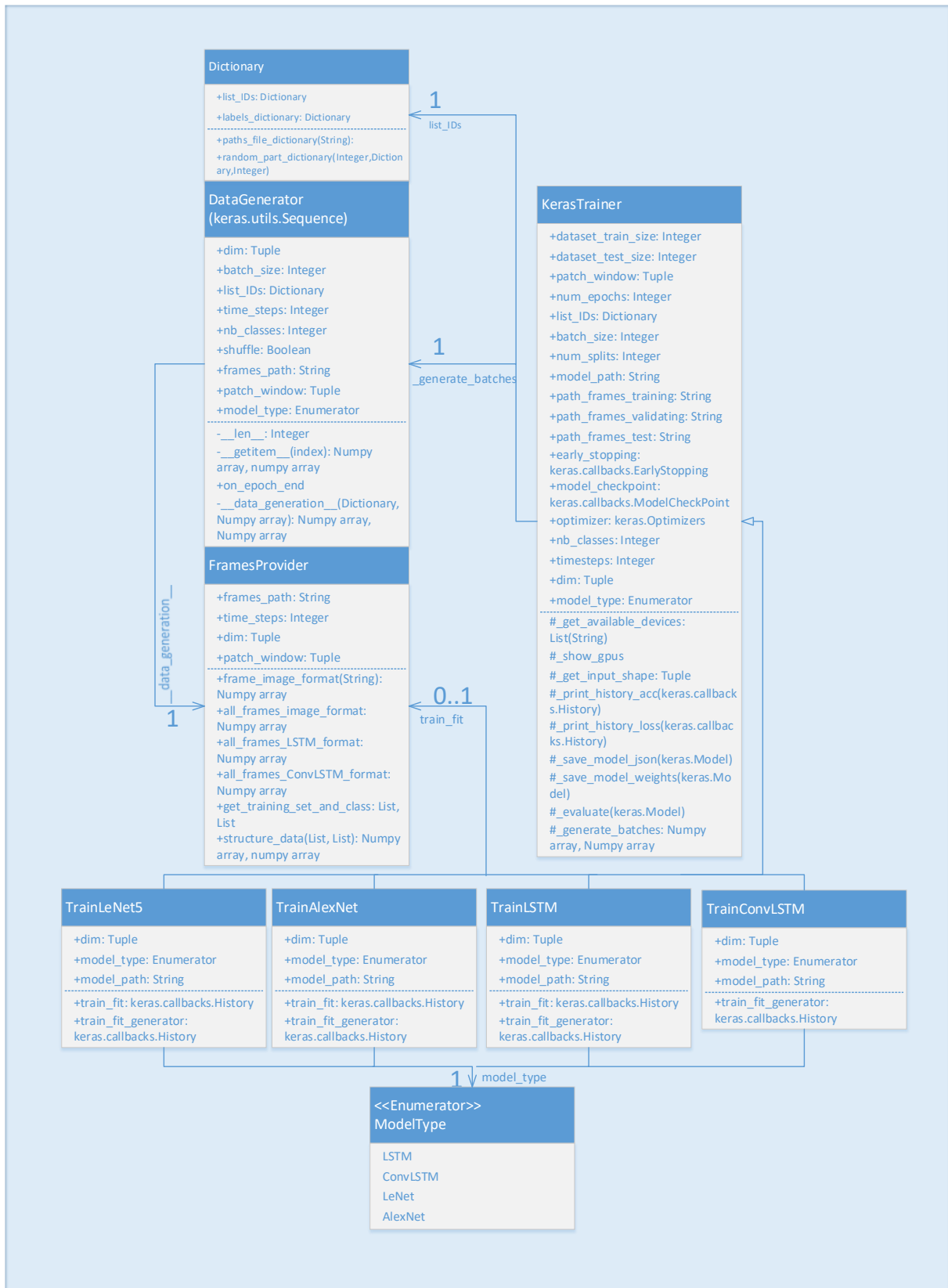


Figure 85: Trainers UML, own elaboration



Figure 86: VideoProvider UML, own elaboration

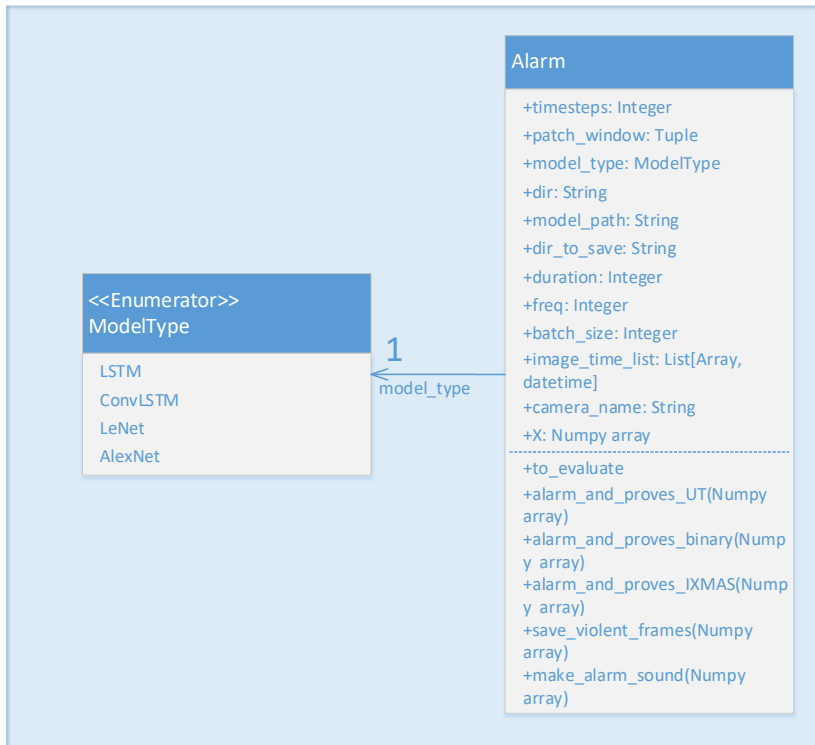


Figure 87\_Alarm System UML, own elaboration

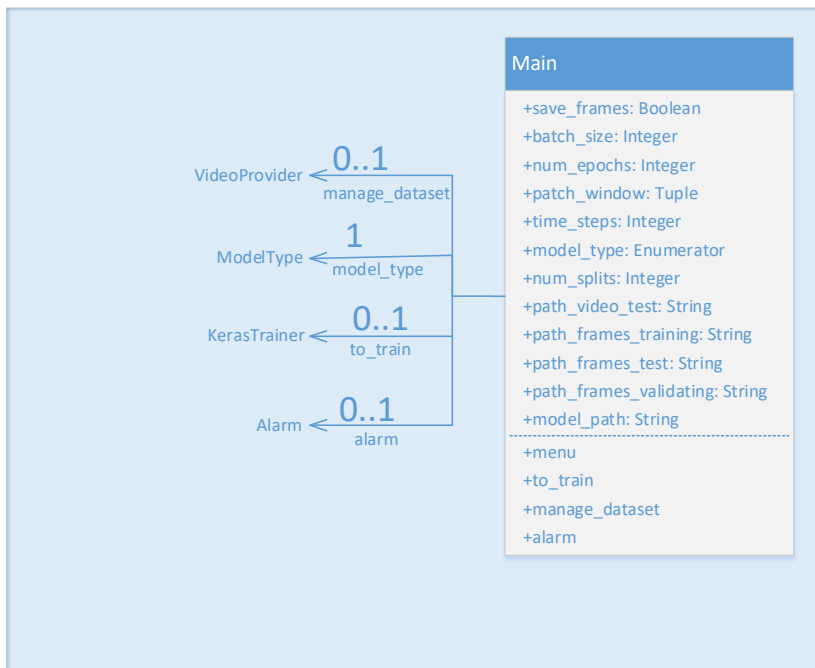


Figure 88: Menu UML, own elaboration



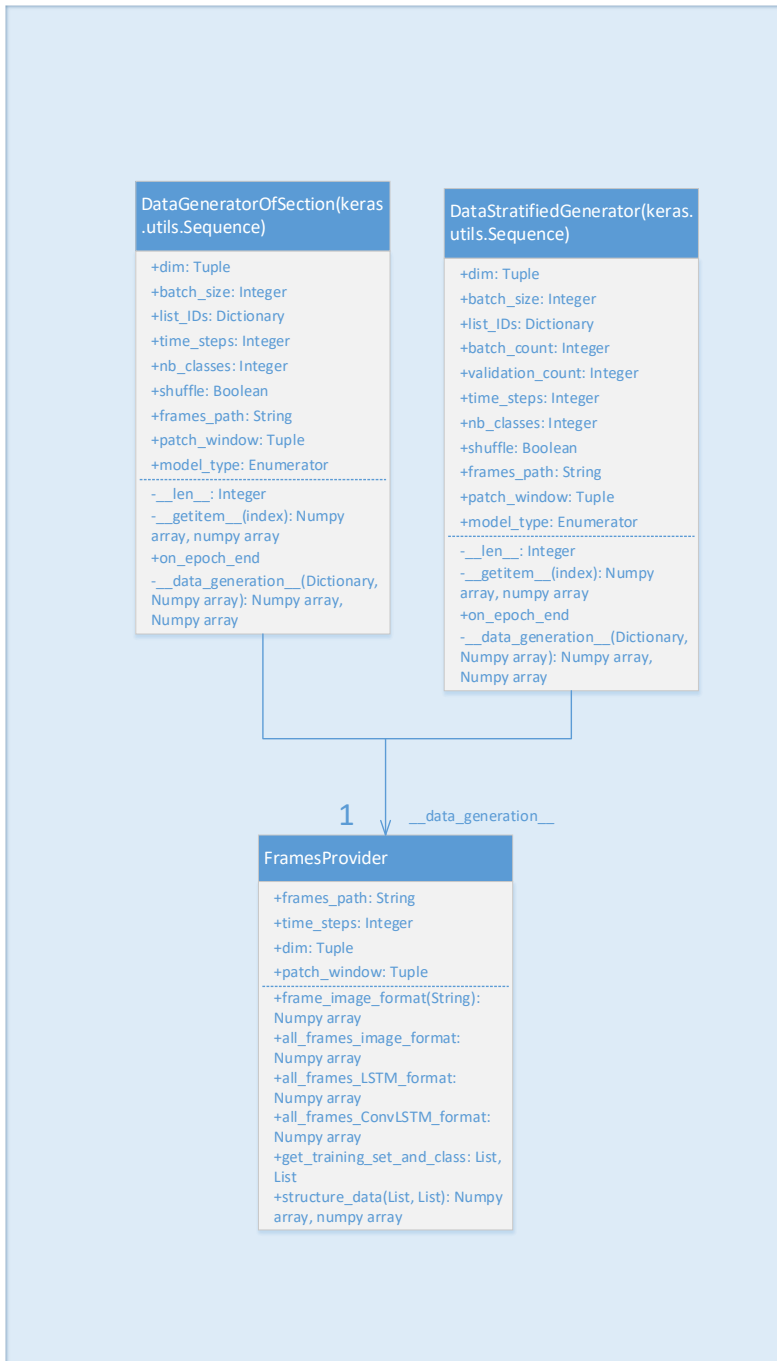


Figure 89: Other generators UML, own elaboration