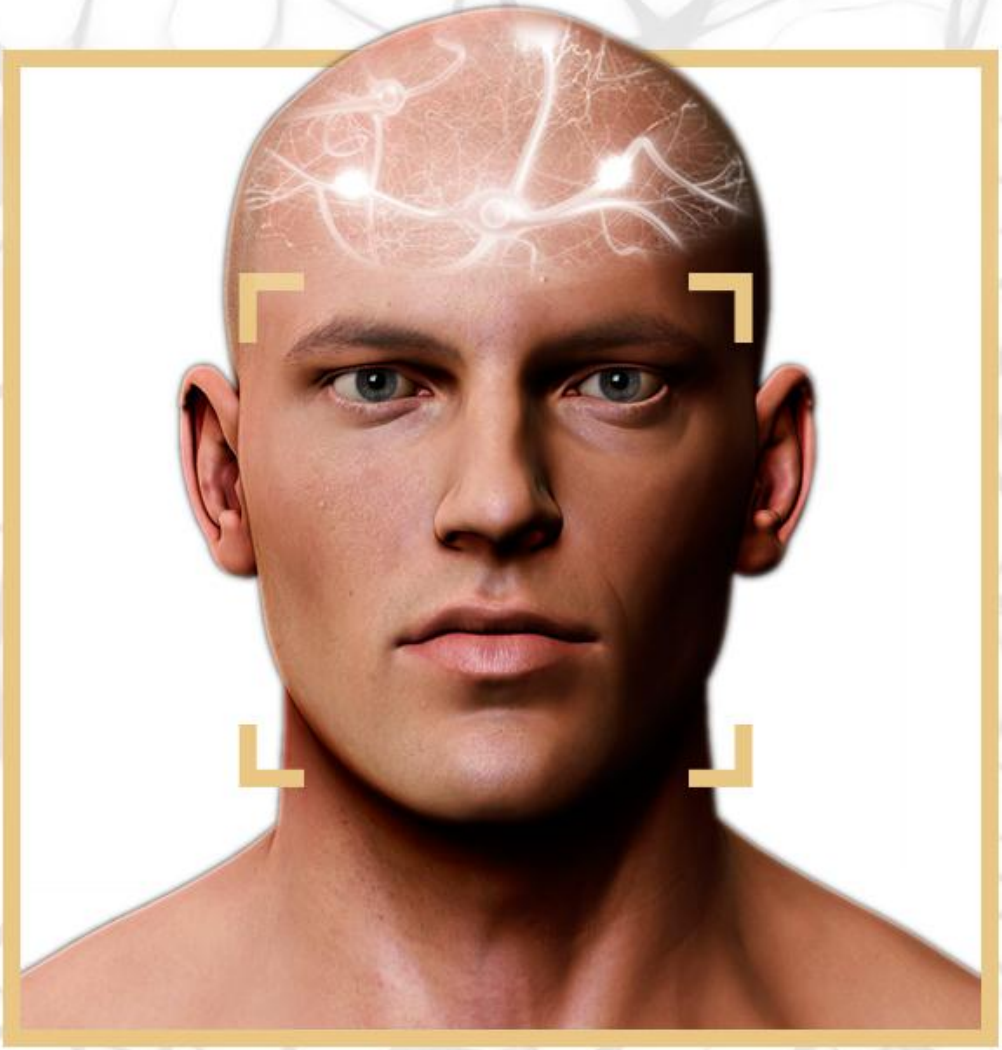


André Filipe Pereira Brás

GESTURE RECOGNITION USING DEEP NEURAL NETWORKS

Master thesis in Mechanical Engineering
in the specialty of Production and Project

July / 2017

· U C ·

Universidade de Coimbra

• U  C •

FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

DEPARTAMENTO DE
ENGENHARIA MECÂNICA

Gesture recognition using deep neural networks

Submitted in Partial Fulfilment of the Requirements for the Degree of Master in Mechanical Engineering in the speciality of Production and Project

Reconhecimento de gestos usando redes neuronais profundas

Author
André Filipe Pereira Brás

Advisor
Professor Pedro Mariano Simões Neto

Jury

President	**Professor Doutor Cristóvão Silva** **Professor Auxiliar da Universidade de Coimbra**
Vowel	**Doutor Nuno Alberto Marques Mendes** **Investigador Auxiliar da Universidade de Coimbra**
Advisor	**Professor Doutor Pedro Mariano Simões Neto** **Professor Auxiliar da Universidade de Coimbra**

Coimbra, July, 2017

ACKNOWLEDGEMENTS

During the last 5 years as student of Mechanical Engineering at University of Coimbra, I worked toughly to reach this final stage. Thus, this dissertation is the climax of all the effort and ambition presented during the course. From the beginning of the hike, I was supported by many people, which collaborated with me without any reluctance. Therefore, I cannot leave without saying a few words of gratitude.

To Mr. Prof. Dr. Pedro Mariano Simões Neto, advisor of this dissertation, I express my sincere gratitude for all the time kindly dispensed, the desire to transmit all the knowledge about the dissertation, but also for his friendship, devotion and willingness.

To all my colleagues in the laboratory, for their patience and disposition to help with any questions and for all the advices that helped to improve this work.

To all my friends with whom I shared these years and with whom I lived experiences and memories that I will never forget. Also, a word to the friends outside of the University who unconditionally encouraged me.

Lastly, the acknowledgements to my strongest pillars throughout this journey. To my parents, my brother and my dear, I have to thank you for all the support and courage given, for allowing me a better future and for always believing in me.

Abstract

This dissertation had as the main goal the development of a method to perform gesture segmentation and recognition. The research was motivated by the significance of human action and gesture recognition in real world applications, such as Human-Machine Interaction (HMI) and sign language understanding. Furthermore, it is thought that the current state of the art can be improved, since this is an area of research in continuous developing, with new methods and ideas emerging frequently.

The gesture segmentation involved a set of handcrafted features extracted from 3D skeleton data, which are suited to characterize each frame of any video sequence, and an Artificial Neural Network (ANN) to distinguish resting moments from periods of activity. For the gesture recognition, 3 different models were developed. The recognition using the handcrafted features and a sliding window, which gathers information along the time dimension, was the first approach. Furthermore, the combination of several sliding windows in order to reach the influence of different temporal scales was also experienced. Lastly, all the handcrafted features were discarded and a Convolutional Neural Network (CNN) was used with the aim to automatically extract the most important features and representations from images.

All the methods were tested in *2014 Looking At People Challenge*'s data set and the best one achieved a Jaccard index of 0.71. The performance is almost on pair with that of some of the state of the art techniques.

Keywords Machine learning, Deep learning, Artificial Neural Networks, Convolutional Neural Networks, Gesture recognition, Pose descriptor.

Resumo

Esta dissertação teve como principal objetivo o desenvolvimento de um método para realizar segmentação e reconhecimento de gestos. A pesquisa foi motivada pela importância do reconhecimento de ações e gestos humanos em aplicações do mundo real, como a Interação Homem-Máquina e a compreensão de linguagem gestual. Além disso, pensa-se que o estado da arte atual pode ser melhorado, já que esta é uma área de pesquisa em desenvolvimento contínuo, com novos métodos e ideias surgindo frequentemente.

A segmentação dos gestos envolveu um conjunto de características artesanais extraídas dos dados 3D do esqueleto, as quais são adequadas para representar cada *frame* de qualquer sequência de vídeo, e uma Rede Neuronal Artificial para distinguir momentos de descanso de períodos de atividade. Para o reconhecimento de gestos, foram desenvolvidos 3 modelos diferentes. O reconhecimento usando as características artesanais e uma janela deslizante, que junta informação ao longo da dimensão temporal, foi a primeira abordagem. Além disso, a combinação de várias janelas deslizantes com o intuito de obter a influência de diferentes escalas temporais também foi experimentada. Por último, todas as características artesanais foram descartadas e uma Rede Neuronal Convolucional foi usada com o objetivo de extrair automaticamente as características e as representações mais importantes a partir de imagens.

Todos os métodos foram testados no conjunto de dados do concurso *2014 Looking At People* e o melhor alcançou um índice de Jaccard de 0.71. O desempenho é quase equivalente ao de algumas técnicas do estado da arte.

Palavras-chave: Aprendizagem de máquina, Aprendizagem profunda, Redes Neuronais Artificiais, Redes Neuronais Convolucionais, Reconhecimento de gestos, Descritor de pose.

Contents

LIST OF FIGURES

LIST OF TABLES

SYMBOLOGY AND ACRONYMS

Symbology

Δ – Margin underlying to the definition of the Multiclass SVM loss

α – Learning rate

$\alpha^{(i,j,k)}$ – Inclination angle defined by all joints in the superscript

$\beta^{(i,j,k)}$ – Azimuth angle defined by all joints in the superscript

$\gamma^{(i)}$ – Bending angle for the i-th joint

$\delta \boldsymbol{p}$ – Joints' velocity in the coordinate system solidary with the body

$\delta^2 \boldsymbol{p}$ – Joints' acceleration in the coordinate system solidary with the body

λ – Regularization strength

$\rho^{(i,j)}$ – Pairwise distance between the joints in the superscript

$A_{s,n}$ – Ground truth of the gesture n at video sequence s

$B_{s,n}$ – Prediction of the gesture n at video sequence s

F – Receptive field of a neuron in a convolution layer

$J_{s,n}$ – Jaccard index of the gesture n at video sequence s

K – Number of different classes

L – Loss evaluated for all training examples

L_i – Output of the loss function for the i-th training example

N – Number of images in the training set

P – Zero-padding, which is a hyperparameter of the convolution layers

R – Regularization penalty to the output of the loss function

S – Stride, which is a hyperparameter of the convolution layers

$b^{(i-1,i)}$ – Average "bone" length that links the joints in the superscript

$cons_{\mathrm{W}}$ – Minimum number of consecutive windows to detect a gesture

l_{DP} – Number of descriptors included in each dynamic pose

lim_{SL} – Segment's length above which is assumed it contains at least 2 gestures

m – Momentum coefficient

min_{GL} – Gesture's length below which the data have to be resized

min_{W} – Minimum number of windows that has to be applied to each gesture

n – Gesture category

p – Probability with which a neuron is kept active

s – Video sequence

s_{DP} – Temporal step between the sampled descriptors to form a dynamic pose

s_{W} – Temporal step that commands the progress of the sliding window

$threshold_1$ – The limit used for segments whose length is lower than lim_{SL}

$threshold_2$ – The limit used for segments whose length is at least equal to lim_{SL}

w_{i} – Width of the input volume

w_{o} – Width of the output volume

y_i – Ground truth label for the i-th training example

V – Matrix initialized with zeros that symbols the velocity of the update

W – Weights matrix shared by all neurons in a convolution layer

b – Bias vector for all activation maps in a convolution layer

$\boldsymbol{p}$ – Normalized joints' position in the coordinate system solidary with the body

$\boldsymbol{p}_{\mathrm{raw}}$ – Raw joints' position in the coordinate system solidary with the Kinect

s – Vector with the classes' scores

$\boldsymbol{u}_{\mathrm{x}}$ – Basis' vector aligned with the spine

$\boldsymbol{u}_{\mathrm{y}}$ – Basis' vector approximately parallel to the shoulder line

$\boldsymbol{u}_{\mathrm{z}}$ – Basis' vector perpendicular to the torso

$\boldsymbol{v}_1$ – Projection of the vector $\boldsymbol{u}_{\mathrm{x}}$ on the plane perpendicular to the orientation of the first "bone"

$\boldsymbol{v}_2$ – Projection of the second "bone" on the plane perpendicular to the orientation of the first "bone"

$\nabla\mathbf{L}$ – Loss function's gradient

Acronyms

ANN - Artificial Neural Network

CNN - Convolutional Neural Network

DBN – Deep Belief Network

GPU – Graphics Process Unit

HMI – Human-Machine Interaction

HOG – Histograms of Oriented Gradients

ILSVRC – *ImageNet Large Scale Visual Recognition Challenge*

LAP – Looking At People

LSTM – Long Short-Term Memory

MLP – Multi-Layer Perceptron

PCA – Principal Component Analysis

ReLU – Rectified Linear Unit

RGB – Red Green Blue

RGB-D – Red Green Blue – Depth

RNN – Recurrent Neural Network

SCG – Scaled Conjugate Gradient

SGD – Stochastic Gradient Descent

SVM – Support Vector Machine

1. INTRODUCTION

1.1. Motivation

In recent years, human action and gesture recognition attracts increasing attention of researchers, playing a significant role in areas such as video surveillance, robotics, Human-Machine Interaction, user interface design and multimedia video retrieval. In fact, gesture recognition is the focus of this work, since it is a central problem in the rapidly growing field of HMI. The survey published by (Maurtua, 2015) shows that gestures are the second choice of industry workers to communicate with collaborative robots, right after the standard pushbutton, which can be problematic when several commands are needed. Another option is voice-based communication, which also may not be a good choice since the industry environment is often very noisy.

Machine learning can be defined as the science of building hardware or software that can achieve tasks by learning from labelled examples. Given new inputs, a trained machine can make predictions about the unknown output. Conventional machine learning techniques were limited in their ability to process natural data in their raw form. For decades, constructing a pattern recognition or a machine learning system required careful engineering and considerable domain expertise to design a feature extractor. This tool should be able to make suitable representations of the raw data from which the learning subsystem, often a classifier, could detect patterns in the input (LeCun, Bengio and Hinton, 2015).

According to the aforementioned, previous works on video-based action recognition focused mainly on adapting handcrafted features. A descriptor for holistic representations of human actions, regarding a video sequence as a whole with spatio-temporal features directly extracted from it was presented in (Shao *et al.*, 2014). The correlation between sequential poses in an action to perform its recognition by combining the advantages of both local and global representations was explored in (Wu and Shao, 2013). Some of the most popular feature descriptors are Cuboids (Dollár *et al.*, 2005), Histograms of Oriented Gradients (HOG) (Laptev and Lindeberg, 2006) and HOG 3D (Kläser, Marszalek and Schmid, 2008). However, the very high-dimensional features usually require the use of dimensionality reduction methods, such as Principal Component Analysis

(PCA), to make them computationally feasible. Furthermore, as discussed by (Wang *et al.*, 2009), the descriptors' performance is data set dependent and no general handcrafted feature outperforms all others existing. For these reasons, there has been a growing interest in extracting more robust and discriminative features through advanced machine learning. Accordingly, it can be interesting to make now an introduction to the most common neural networks.

A standard neural network consists of many simple and connected units, also known as neurons, which produce a sequence of values, commonly named activations. Input neurons get activated through sensors perceiving the environment, while other neurons get activated through weighted connections from the previously active neurons (details in CONVOLUTIONAL NEURAL NETWORKS section). Then, as (Schmidhuber, 2015) described, learning is related with the search for weights that make the neural network exhibit the desired behaviour, such as recognizing human gestures. As a result of progress and development, deep learning arrived and, following (LeCun, Bengio and Hinton, 2015), it allows computational models composed by multiple processing layers to learn representations of data with multiple levels of abstraction, building high-level features from low-level ones. Particularly, Convolutional Neural Networks (Lecun *et al.*, 1998) are a type of deep models in which trainable filters and local neighbourhood pooling operations are applied alternatingly on the raw input images, resulting in a sequence of increasingly complex features. As a consequence of their success (Krizhevsky, Sutskever and Hinton, 2012; Ciresan, Meier and Schmidhuber, 2012; Zeiler and Fergus, 2014), CNNs-based models are in use by various industry leaders like Google, Facebook and Amazon. Recently, researchers at Google applied CNNs on video data (Karpathy *et al.*, 2014). Furthermore, it is expected that Convolutional Neural Networks will have diverse applications in technology, including autonomous mobile robots and self-driving cars (Hadsell *et al.*, 2009; Farabet *et al.*, 2012). It is important to realise that the key aspect of deep learning is that those layers of features are not designed by human engineers; instead, they are learned from data using a learning procedure.

Since the recent resurgence of neural networks invoked by (Hinton, Osindero and Teh, 2006), deep neural architectures have effectively become an approach to execute automated extraction of features. Consequently, deep learning is helping to achieve major advances in solving problems that remained unsurpassed for many years. (Schmidhuber,

2015) wrote an historical survey summarizing relevant works. From this overview, it is possible to realise that these models have been successfully applied to a myriad of different domains: image classification (Krizhevsky, Sutskever and Hinton, 2012), handwritten digits and traffic signs recognition (Ciresan, Meier and Schmidhuber, 2012), Human-Machine Interaction (Cecotti and Graser, 2011), human action recognition in surveillance videos (Ji *et al.*, 2013), speech classification (Mohamed, Dahl and Hinton, 2012), natural language understanding (Collobert *et al.*, 2011), among others.

The invention of the low-cost Microsoft Kinect opened up new opportunities to solve fundamental problems in computer vision. Accordingly, there has been considerable interest in developing methods for preprocessing, object tracking and recognition, human activity analysis, hand gesture analysis and indoor 3D mapping (Han *et al.*, 2013). Although gesture recognition based on RGB (Red Green Blue) or 3D skeletal data may seem trivial, there are some factors that difficult the task. Notice that the wide diversity of backgrounds, differences in lighting, dissimilar view angles and the huge variability with which the gestures are made by each subject, including the infinitely many kinds of out-of-vocabulary motion, are considerable obstacles in robust gesture recognition. On the other hand, the segmentation of different gestures also complicates the task. In practice, segmentation is as relevant as the recognition, but it is a recurrently neglected aspect of the contemporary research, since it is assumed that pre-segmented sequences are available.

This dissertation aims to address some of the aforementioned issues. It focuses on labelling acyclic video sequences, i.e. video sequences that are non-repetitive. Firstly, there is an approach to perform offline segmentation, which is based on a set of handcrafted features built from 3D skeletal data and used to train an Artificial Neural Network. After that, in order to accomplish the classification, three distinct concepts are tested. While in the first try, a single sliding window is applied, in the second one there are more windows to get information about different temporal scales. Both approaches are based on handcrafted features. The third and last approach introduces deep learning, since a CNN is used to perform automatic learning of representations from RGB data.

The remainder of this work is organised as follows. Still in the INTRODUCTION, the work related with action and gesture recognition is revised and the data set used along the text is presented. CONVOLUTIONAL NEURAL NETWORKS completes one of the goals of this work, since it does a global but easy understanding review

of CNNs, including the learning process. GESTURE SEGMENTATION details the procedure of collecting data and training the ANN to perform the segmentation task. GESTURE CLASSIFICATION introduces the postulates behind each classification model and shows the corresponding results. CONCLUSION finishes the work and gives hints about what it is possible to do in the future.

1.2. Related Work

As introduced above, traditional approaches to action and gesture recognition typically include spatio-temporal engineered descriptors, which are followed by the classification process. Even several top winning methods of the *2014 Looking At People Challenge* (Escalera *et al.*, 2015) require handcrafted features for either skeletal data, RGB-D (Red Green Blue – Depth) data, or both. For instance, (Neverova *et al.*, 2015a) proposed a moving pose descriptor consisting in subsets of features drawn from skeleton data. This well-defined descriptor will be the main guideline of the initial phase of this work. Classified in second place, (Monnier, German and Ost, 2015) proposed 4 types of features for skeleton data. Additionally, they also used the HOG descriptor for RGB-D images. This combination of skeletal information with the HOG descriptor was already used by (Chen and Koskela, 2013) to perform online gesture recognition. Furthermore, (Peng *et al.*, 2015) adopted handcrafted features based on dense trajectories (Wang *et al.*, 2013) for the RGB data.

Convolutional Neural Networks have been primarily applied on bidimensional images. The approach proposed by (LeCun *et al.*, 1989) was successfully applied to the recognition of handwritten zip code digits. Notice that, at that time, the name of this type of networks was yet unknown. By the late 1990s this system was reading over 10% of all the cheques in the United States. The report published by (Krizhevsky, Sutskever and Hinton, 2012) was an important breakthrough due to the use of CNNs to almost halve the error rate for object recognition. Therefore, it acted like a trigger and, consequently, it precipitated the rapid adoption of deep learning by the computer vision community. Bo Yang *et al.* proposed a systematic feature learning method for the human action recognition problem (Bo Yang *et al.*, 2015). They used a CNN to automate feature learning from raw inputs, which were time series signals acquired from a set of body-worn inertial sensors. Furthermore, they mutually enhanced feature learning and classification by unifying them in one model. The present

work explores the use of CNNs for human gesture recognition in videos. A simple approach in this direction is to treat video frames as still images and apply CNNs to recognize actions at the individual frame level. However, to effectively incorporate motion information encoded in multiple contiguous frames, a specific preprocessing will be done. Moreover, 3D CNNs, which are introduced below would be another interesting approach.

There have been a few more works exploring deep learning for action recognition in videos (Yang *et al.*, 2009; Taylor *et al.*, 2010; Ji *et al.*, 2013; Pigou *et al.*, 2015; Wu *et al.*, 2016). For example, (Ji *et al.*, 2013) used 3D Convolutional Neural Networks to recognize human actions in the airport surveillance videos. Their model extracts features from both the spatial and the temporal dimensions by performing 3D convolutions, thereby capturing the motion information encoded in multiple adjacent frames. To further boost the performance, they proposed regularizing the outputs with high-level features and combining the predictions of a variety of different models. A method to perform simultaneous, multimodal gesture segmentation and recognition is presented in (Wu *et al.*, 2016). A Deep Belief Network (DBN) and a 3D CNN are adjusted to manage skeletal and RGB-D data, respectively. (Taylor *et al.*, 2010) also explored 3D CNNs for learning spatio-temporal features, which help to understand video data.

The progress described above shows a growing trend, since various fundamental architectures have been proposed in the context of motion analysis for learning representations directly from data, as opposed to handcrafting. In fact, the computer vision community is devoting its efforts to deep learning, in order to automate the building of features. Besides that, it has been common the development of deep architectures for multimodal data, which have the chance of obtaining more information about the action.

When the tasks involve sequential inputs, such as the current topic of human action and gesture recognition, Recurrent Neural Networks (RNNs) is often applied with success. These networks process an input sequence with one element at a time, maintaining in their hidden units a state vector that implicitly contains information about the history of all the past elements of the sequence. Essential to that success is the use of Long Short-Term Memories (LSTMs) (Hochreiter and Schmidhuber, 1997), a special kind of RNNs, which works, for many tasks, better than the conventional version. A deep learning model composed by convolutional and LSTM recurrent layers, which is capable of automatically learning feature representations and modelling the temporal dependencies between them, is

presented in (Ordóñez and Roggen, 2016). They demonstrated that this method is suitable for human activity recognition from wearable sensor data with minimal preprocessing using it on two families of motions: periodic activities, such as modes of locomotion, and sporadic activities, like gestures. A method that explicitly models the video as an ordered sequence of frames was proposed and evaluated with success in (Ng *et al.*, 2015). For that purpose, they employed a recurrent neural network that uses LSTM cells, which are connected to the output of the underlying feature extractor. To conclude, the results above revealed that, in video domain, CNNs and LSTMs are suitable to combine temporal information in subsequent video frames and, hence, to enable better video classification.

1.3. *2014 Looking At People Challenge*'s Data Set

ChaLearn is an organization with vast experience about coordination of academic challenges in several interrelated fields, including machine learning. Looking At People (LAP) is a division of ChaLearn and a challenging area of research that deals with the problem of automatically recognizing people in images, detecting and describing body parts, inferring their spatial configuration and performing action and gesture recognition from still images or image sequences (Escalera *et al.*, 2017).

When ChaLearn LAP organizes new events, they are motivated both by academic interest and by potential applications, such as TV production, home entertainment, education purposes, surveillance and security, improved life quality by automatic artificial assistance, among others. The ChaLearn LAP platform, which contains all information and resources from previous and current events, including programs, codes and data, is introduced in (Escalera *et al.*, 2017).

1.3.1. Competition Data

A particular case of the several organized contests is *2014 Looking At People Challenge*. ChaLearn prepared in 2014 three parallel challenge tracks in human pose recovery, action/interaction spotting and gesture recognition. The third track's data set includes nearly 14,000 gestures drawn from a vocabulary of 20 Italian sign gesture categories, but also contains multiple unlabelled gestures to baffle. Data products embrace colour and depth video, segmentation masks and skeleton joints information produced by a

Kinect sensor. Figure 1.1 illustrates different visual modalities for a single video frame. The challenge's data set is split into training, validation and testing sets. Originally, only the training data was accompanied by a ground truth label for each gesture, as well as information about its starting and ending points. Even though the ground truth label for validation and test sets had already been released, it only will be used to evaluate the models.

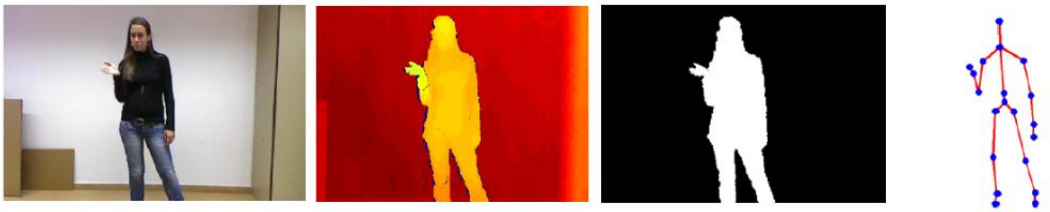


Figure 1.1. Different modalities of data delivered for the *2014 Looking At People Challenge*'s third track (Escalera *et al.*, 2015).

The emphasis of *2014 Looking At People Challenge*'s third track is on multi-modal automatic learning of a set of 20 gestures performed by several different users, with the aim of performing continuous gesture spotting, independent of the user. Thereby, this is one of the reasons that make this data set an interesting one. During this challenge, the state of the art was significantly improved since solutions based in deep learning obtained the best results.

1.3.2. Evaluation Criteria

Recognition performance is evaluated using the Jaccard index, which relies on a frame-by-frame prediction accuracy. In this sense, for each video sequence, the Jaccard index, $J_{s,n}$, of each one of the 20 labelled gesture categories is defined as follows:

$$J_{s,n} = \frac{A_{s,n} \cap B_{s,n}}{A_{s,n} \cup B_{s,n}}, \tag{1.1}$$

where $A_{s,n}$ is the ground truth of gesture n at video sequence s and $B_{s,n}$ is the obtained prediction for such gesture in the same sequence. Here, $A_{s,n}$ and $B_{s,n}$ are binary vectors where the frames in which the n-th gesture is being performed are marked with 1. The final score is the mean Jaccard index among all gesture categories for all sequences.

In case of false positives (e.g. inferring a gesture not labelled in the ground truth), the Jaccard index is zero for that particular prediction and it will count in the mean Jaccard

index computation. In other words, n is equal to the intersection of gesture categories appearing in the ground truth and in the predictions.

Finally, it is important to point out that, being defined at the frame level, the Jaccard index can vary due to deviations of the segmentation (both in the ground truth and prediction) at gesture boundaries. Often, these small variations can be irrelevant from an application viewpoint and then the Jaccard index can be considered very severe for the quality of the model.

2. CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks have been some of the most influential innovations in the field of computer vision. They were used to win *ImageNet Large Scale Visual Recognition Challenge 2012* (ILSVRC 2012) and hence CNNs grew to prominence (Krizhevsky, Sutskever and Hinton, 2012). Ever since then, a host of companies have been using deep learning at the core of their services. Facebook uses neural networks for their automatic tagging algorithms, Google for their photo search and Amazon for their product recommendations.

Despite of all CNNs' applications, their classic and arguably most popular use case is image processing, in particular image classification, which is the task of taking an input image from a fixed set of categories and assigning a label or a probability distribution over labels that best describe the image. The image classification problem can be divided into 2 main parts. The input, which is called the training data, consists of a set of N images, each one labelled with one of the K different classes. Then, this training data is used by the network to learn with what every one of the classes looks like. The network is made up of multiple layers that convert a 3-dimensional input volume to a 3-dimensional output volume. Each transformation is performed using some differentiable function that may or may not have parameters. In the end, the quality of the classifier is evaluated comparing the true labels with the predicted ones for a new set of images, the test data, that it has never seen before. As an evaluation criterion, it is common to use the accuracy, which measures the fraction of predictions that are correct.

The following sections have the purpose to facilitate the understanding of what every layer in a CNN does. Then it is also an intention of this chapter to explain how the learning process, which is responsible for the parameters update, works.

2.1. Input Layer

CNNs' architectures make the explicit assumption that their inputs are images. To a computer, an image is represented as a large 3-dimensional array of brightness values that range from 0 (black) to 255 (white). Commonly, the array's height and width are called

spatial dimensions. The remaining dimension is termed depth and it corresponds to the image's number of channels. The word *depth* here refers to the third dimension of the input image, not to the depth of a full neural network, which is the total number of layers in the network. As an example, if the number of channels is 3, the input is a simple RGB image. It is also possible to use RGB-D images and, in this case, the input image has 4 channels.

2.2. Fully Connected Layer

A fully connected layer is not close of being the first layer in a CNN. However, for historical and logic reasons, it seems reasonable to introduce it now. As it will be explained in more detail below, the name of this type of layers is due to the connections between its neurons and the neurons in the previous layer.

Artificial Neural Networks (Lippmann, 1987) are computation models which try to mimic biological nervous system. They were developed for the first time few decades ago, but todays increased availability of computational power and new training techniques allowed the development of new applications. An ANN takes the inputs in the first layer and transforms them through a series of fully connected layers. Each layer is made up of a set of neurons that are fully and weightily connected to all neurons in the previous layer. Beyond this, the neurons within a single layer function independently and do not share any connections. Each neuron performs a dot product between its inputs and weights and adds the bias. Then it is applied the non-linearity, also called the activation function, whose details will be addressed afterward. The general model of an ANN is in Figure 2.1.

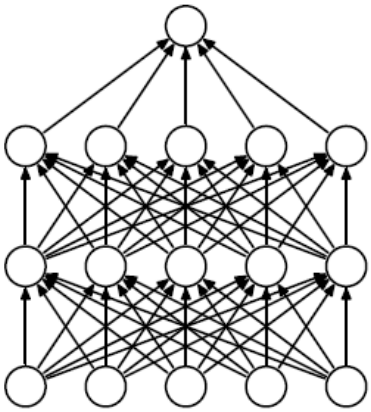


Figure 2.1. A 3-layer neural network with 5 inputs. There are 2 hidden layers of 5 neurons each and an output layer with 1 neuron. So this ANN has 11 neurons, 55 weights and 11 biases (Srivastava *et al.*, 2014).

The last fully connected layer is called the output layer and its size should be equal to the number of different classes, K. Unlike all layers in an ANN, the neurons of this layer have not an activation function because its output is usually taken to represent the classes' scores. The layers between the input and the output layers are called hidden layers and its sizes are hyperparameters of the network.

2.3. Convolution Layer

Convolution layers are the core building block of CNNs. When dealing with high-dimensional inputs, such as images, the full connectivity of neurons in ANNs is wasteful and the huge number of parameters would quickly lead to overfitting[1]. Instead, each neuron is only connected to a small region of the previous layer. The spatial extent of this connectivity is a hyperparameter called the receptive field, F, of the neuron. The extent of the connectivity along the depth axis is always equal to the depth of the input volume. Beyond this, convolution layers are very similar to ordinary fully connected layers: they are made up of neurons that receive some inputs, perform a dot product, add the bias and optionally follow the output with a non-linearity.

The convolution layers' weights are sets of learnable filters. Every filter is small spatially, but extends through the full depth of the input volume. Note that the spatial dimensions of the filter are equivalent to the receptive field. During the forward pass, each filter slides – more precisely, convolves – across the width and height of the input volume and, at any position, it computes dot products between the weights of the filter and the input values. Each filter produces a 2-dimensional activation map, but it is usual to have a set of multiple filters in each convolutional layer. The number of filters and hence the number of activation maps in a convolutional layer is a hyperparameter and it corresponds to the depth of the output volume. Each activation map is added with a different value called bias that influences the output scores without interacting with the data. In the future, when the training is finished, filters will activate if they see particular features, such as edges of some orientation, blobs of some colour or eventually wheel-like patterns. For this reason, filters like those depicted in Figure 2.2 can be thought of as feature identifiers.

[1] Overfitting occurs when a model with high capacity fits the noise in the data instead of the underlying relationship. In other words, if overfitting is a real problem, the network's weights are so tuned to the training examples given to them that the network does not work well when new examples are given.

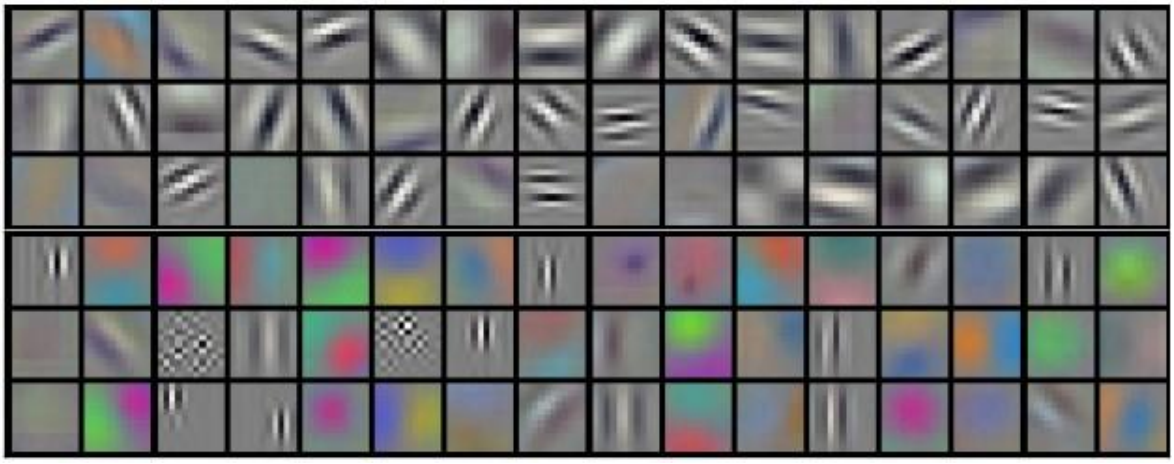


Figure 2.2. 96 filters of size 11×11×3 learned by the first convolutional layer of a CNN on the 227×227×3 input images. Each filter is shared by the 55×55 neurons in each activation map. It is easy to realise that the filters in the first convolution layer are designed to detect low level features such as edges, curves and blobs of colour (Krizhevsky, Sutskever and Hinton, 2012).

Besides the receptive field, there are 2 more hyperparameters that influence decisively the output's spatial dimensions, such as the stride and zero-padding. The stride, S, controls the number of pixels that the filter convolves at each step. When the stride is 2, then the filters jump 2 pixels at a time and the output volume will be spatially smaller than the input one. Normally, programmers increase the stride if they want receptive fields to overlap less and if they want smaller spatial dimensions. Sometimes, it will be convenient to pad the input volume with zeros around the border. The size of this zero-padding, P, is a hyperparameter. The nice feature of zero-padding is that it allows to control the spatial size of the output volume. As example, the output volume's width, w_{o}, can be computed as a function of the input volume's width, w_{i}, and the hyperparameters defined above:

$$w_{\mathrm{o}} = (w_{\mathrm{i}} - F + 2{\times}P){\times}S^{-1} + 1. \tag{2.1}$$

Parameter sharing scheme is used in convolution layers to control the number of parameters. This architecture works reasonably because, if a set of weights is useful to detect some feature at a spatial position, then it should also be useful to detect it at a different position. In other words, the neurons in a single activation map are restrained to use the same weights and bias. This is why it is common to refer the set of weights as a filter that is convolved around the input volume.

2.4. Activation Function

The activation function – also known as non-linearity – is critical computationally because if it is left out, the parameter matrices can be collapsed to a single one and therefore the neural network's output would be a linear function of the input. Every activation function takes a single number called activation as input and performs a certain fixed mathematical operation on it. There are several activation functions that can be encountered in practice.

Historically, a common choice for the non-linearity is the sigmoid function, since it takes an input and smashes it to range between zero and the unity. However, it has recently fallen out of favour and it is rarely used because it owns 2 major drawbacks. On the one hand, the gradient is almost null when the activation function's output is close to zero or close to the unity, which hampers the learning process. On the other hand, the sigmoid's outputs are not zero-centered. This is undesirable since neurons in later layers of a neural network would be receiving data that is not zero-centered.

Other common activation function is the hyperbolic tangent, which squeezes an input number to range between -1 and 1. It is always preferred to the sigmoid non-linearity since its output is zero-centered. Nonetheless, the null gradients are still a problem.

The Rectified Linear Unit (ReLU) as activation function has become very popular in the last few years. It simply thresholds all activations that are below zero to zero. In other words, if the input value is negative, it becomes zero and remains unchanged otherwise. Mathematically, ReLU looks as follows:

$$f(x) = \max(0, x). \tag{2.2}$$

The convergence of the learning process is 6 times faster with ReLU compared to the hyperbolic tangent (Krizhevsky, Sutskever and Hinton, 2012). Moreover, unlike hyperbolic tangent and sigmoid activation functions, ReLU does not involve expensive operations. Unfortunately, it is possible some neurons never activate across the entire training data set. If this happens, the gradient flowing through the unit will forever be zero from that point on. Leaky ReLU (Maas, Hannun and Ng, 2013) is one attempt to fix this problem. Instead of the function being zero when the activation is negative, it has a small slope.

2.5. Pooling Layer

In CNNs, it is common to periodically insert a pooling layer between convolution layers. Its functions are to progressively reduce the spatial size of the representation, reduce the number of parameters and the amount of computation in the network and also control overfitting. The pooling layer operates independently on every activation map and resizes it spatially.

There are several pooling layers, but max pooling layer is a popular choice. This basically takes a filter and a stride, applies it to the input and returns the maximum number in every sub-region covered by the filter. The most common form of max pooling layer uses square filters of size 2 applied with a stride of 2, downsampling every activation map in the input by 2 along both spatial dimensions and discarding 75% of the activations. This case is depicted in Figure 2.3. Other options for pooling layers are average pooling and L2 norm pooling.

The intuitive reason behind this layer is that once it is known that a specific feature is in the original input volume – there will be a high activation value –, its exact location is not as important as its relative location to the other features.

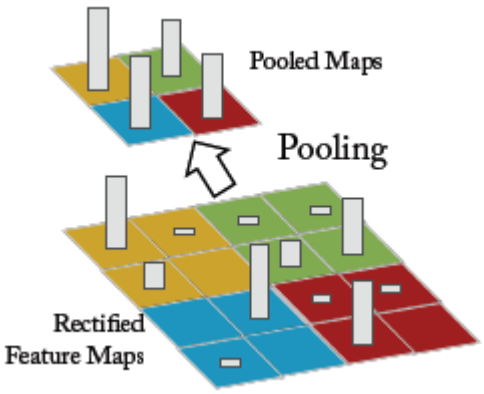


Figure 2.3. An illustration of the max pooling operation: 4 square filters – coloured zones – of size 2 are applied with a stride of 2. Therefore, the activation map is downsampled by 2 along both spatial dimensions (Zeiler and Fergus, 2014).

2.6. Backpropagation

The previous sections described an algorithm parameterized by a set of weights, **W**, and bias, **b**, that determines classes' scores from pixel values. Although data are not manageable, these parameters can be reset so that the predicted classes' scores are consistent with the ground truth labels in the training data. A machine can adjust filter values through a training process called backpropagation. This procedure can be separated into 4 distinct sections, namely the forward pass, the loss function, the backward pass, and the parameters update.

The forward pass was intensively described above since it is the process that allows to get classes' scores from the pixel values of an input image. The other 3 steps will be presented in the following sections.

2.6.1. Loss Function

The agreement between the predicted scores and the ground truth labels is measured with a loss function, sometimes also referred as the cost function. Intuitively, the loss will be high if the previously mentioned agreement is poor and it will be low in the opposite case. To resume, the image classification problem can be shaped as an optimization problem, which is intended to minimize the loss by changing the network's parameters.

There are several ways to define the details of the loss function, such as the well-known Multiclass Support Vector Machine (SVM) loss (Weston and Watkins, 1999) and the Softmax classifier. In the next subsections, the goal is to explore each of them.

2.6.1.1. Multiclass Support Vector Machine Loss

The Multiclass SVM loss takes one particular approach to measure how consistent the predictions on training data are with the ground truth labels. Basically, it is set up to accumulate loss if the correct class's score for each image is not higher than the incorrect classes' scores by some fixed margin Δ. Supposing that **s** is a vector containing the classes' scores, the Multiclass SVM loss for the i-th training image, L_i, is then formalized as follows:

$$L_i = \sum_{j \neq y_i} \max\left(0, s_j - s_{y_i} + \Delta\right). \tag{2.3}$$

In the equation above, y_i is the ground truth for the i-th training image and, hence, s_{y_i} represents the correct class's score. Note that the loss remains unchanged when the correct class's score is greater than incorrect classes' scores by at least Δ.

There is one issue with the loss function presented above, since different sets of parameters can take to similar losses. This means that a set of parameters is not necessarily unique. To remove this ambiguity, it is necessary to encode some preference for a certain set of parameters, which can be done by summing a regularization penalty, R, to the loss function. The most common regularization penalty is the L2 norm that discourages large parameters through a quadratic penalty over all weights:

$$R = \sum_k \sum_l \sum_m W_{klm}^2. \tag{2.4}$$

There are other appealing properties to include the regularization penalty. Since the L2 norm prefers smaller and more diffuse weight vectors, the classifier is encouraged to consider all input values in small amounts rather than a few input values and very strongly. This effect tends to improve the generalization performance of the classifiers on test images and hence it leads to less overfitting. It is possible to use other regularization methods, such as L1 norm, max norm constraints and dropout. The last one is an extremely effective, simple and recently introduced regularization technique by (Srivastava *et al.*, 2014). While training, dropout is implemented by only keeping a neuron active with some probability p or setting it to zero otherwise.

The regularization penalty completes the Multiclass SVM loss, which is thus made up of 2 components: the data loss – which is the average loss L_i over all examples in training set – and the regularization loss. Thereby, the Multiclass SVM loss becomes:

$$L = \frac{1}{N} \times \sum_i L_i + \lambda \times R. \tag{2.5}$$

In this equation, λ is a hyperparameter that weighs the significance of the regularization penalty and, hence, it is termed regularization strength. Since the group $\lambda \times R$ undertakes an important role in preventing overfitting, the hyperparameter λ is like a quantifier of the model generalization. In other words, if the regularization strength is low, the model tends to overfit the data. Otherwise, the regions' thresholds become smoother and the model generalization increases.

2.6.1.2. Softmax Classifier

The other popular choice to quantify the model performance is the Softmax classifier, which has a different loss function. Unlike the Multiclass SVM loss, which treats the inputs as scores for each class, the Softmax classifier gives a slightly more intuitive output and also has a probabilistic interpretation. Concretely, the Softmax classifier interprets its inputs as the unnormalized logarithmic probabilities for each class. So, the probability of the k-th class can be calculated using the Softmax function:

$$f(s_k) = \frac{e^{s_k}}{\sum_j e^{s_j}}. \tag{2.6}$$

Note that the Softmax function takes a vector of arbitrary scores and squashes it to a vector of values between zero and the unity, whose sum is always equal to the unity. The loss function is called the cross-entropy function and it is defined as in the equation (2.7). As before, the full loss for the training set is the mean of L_i over all training examples summed with the regularization penalty.

$$L_i = -ln\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right). \tag{2.7}$$

2.6.2. Backward Pass

In the previous section, 2 different types of the loss function were exploited, which measures the quality of a set of parameters based on how well the induced labels agree with the ground truth ones in the training data. The main goal of this section is to introduce the backward pass, which is the process of understanding how the loss function is influenced by every weight and bias. In other words, it is a way of computing the gradient of the loss function through recursive application of chain rule.

Every neuron in convolution or fully connected layers gets some inputs, namely the outputs of the previous layer and the respective filter's parameters, and compute 2 different things, such as its output value and the partial derivatives of this output value concerning to all of its inputs. Notice that obtaining the output value is as simple as multiplying and adding several numbers, so the partial derivatives are easy to compute. Furthermore, the neurons can do this without being aware of the details of the full network in which they are embedded in.

Once the forward pass is over, during the backward pass the neuron will learn about the loss function's partial derivative with respect to its output value. Then the chain rule says the neuron should take this partial derivative and multiply it by every partial derivative computed before. The values obtained are the loss function's partial derivatives with respect to all neuron's inputs.

The point of this section is that the details of how the backpropagation is performed and the parts in which the full forward function should be partitioned is a matter of convenience. It helps to be aware of which parts of the expression have easy partial derivatives, so that they can be chained together with the least amount of code and effort.

2.6.3. Parameters Update

The neural networks' loss function is usually defined over very high-dimensional spaces, making them difficult to visualize. Therefore, these functions are non-convex, but complex and bumpy terrains. One analogy that can be helpful is to think of a hiker on a hilly terrain trying to reach the bottom. He/she will feel the slope of the hill and step down in the direction that is steepest. Mathematically, this direction is related to the gradient of the loss function, which is just the vector of partial derivatives computed during the backward pass.

Now that it is known how to compute the gradient of the loss function, the procedure of repeatedly evaluating the gradient and then performing a parameter update is called Gradient Descent. This method is currently by far the most common and established way of optimizing neural networks' loss function. Its vanilla version looks as follows:

$$\mathbf{W} = \mathbf{W} - \alpha \times \nabla \mathbf{L}, \tag{2.8}$$

where α is termed step size – also called the learning rate – and $\nabla \mathbf{L}$ contains partial derivatives of the loss function along all dimensions. Assume that $\nabla \mathbf{L}$ and $\mathbf{W}$ have the same dimensions and that ∇L_{klm} is the loss function's partial derivative with respect to the weight W_{klm}. Choosing the learning rate will become one of the most important decisions in training a neural network, since lower values lead to small but confident updates, while larger values in attempt to descend faster can give a higher loss – overstep.

When training data have a huge number of examples, it seems wasteful to compute the loss function over the entire training set to perform only a single parameter

update. A very common approach to address this challenge is to compute the gradient over batches of training data. Therefore, much faster convergence can be achieved by using batch gradients to perform more frequent parameter updates. The extreme case of this is a setting where the batch contains only a single example. This process is called Stochastic Gradient Descent (SGD).

Gradient Descent with Momentum is another approach that almost always enjoys better converge rates on deep neural networks. Initializing the parameters with random numbers is equivalent to setting a particle with zero initial velocity at some location of the hilly terrain. The optimization process can then be seen as equivalent to the process of simulating the particle rolling on the landscape. Gradient Descent with Momentum suggests an update in which the gradient only directly influences the velocity, which in turn has effect on the position. Mathematically, it is formalized as follows:

$$\mathbf{V} = m \times \mathbf{V} - \alpha \times \nabla \mathbf{L}, \tag{2.9}$$

$$\mathbf{W} = \mathbf{W} + \mathbf{V}, \tag{2.10}$$

where $\mathbf{V}$ is a variable that is initialized with zeros and m is a hyperparameter named momentum, whose physical meaning is more consistent with the coefficient of friction. Effectively, this variable shrinks the velocity and reduces the kinetic energy of the system, or otherwise the particle would never come to stop at the bottom of the hill. Gradient Descent has other versions and there are also more methods for optimization in context of deep learning.

2.7. Considerations

As described above, a CNN is a sequence of few distinct layers and every layer transforms a volume of activations to another through a differentiable function. Therefore, CNNs transform the original image, layer by layer, from the original pixel values to the final classes' scores. Only some layers, like convolution and fully connected layers, contain parameters. These parameters are trained with SGD in order to the classes' scores computed by the network be consistent with the images' labels in the training set.

This process of forward pass, loss function, backward pass and parameter update over all input images is generally called an epoch. In other words, the number of epochs

measures how many times every example was seen during the training. The program will repeat the process for a fixed number of epochs over each batch of training images.

An advantage of this approach is that the training data is used to learn the parameters but, once the learning is completed, it is possible to discard the entire training data and to only keep the learned parameters. That is because a new test image can be simply forwarded through the network and classified based on the optimized parameters.

The most common form of a CNN architecture stacks a few convolution layers followed by an activation function, applies a pooling layer and repeats this pattern until the image has been merged spatially to a small size. At some point, it is common to transit to the fully connected layers. The last fully connected layer holds the output, such as the classes' scores.

3. GESTURE SEGMENTATION

Different gestures can be very similar on their initial and final phases, so framewise classification at these periods is often uncertain and therefore erroneous. To prevent these negative effects, a module for gesture detection and segmentation is introduced. Specifically, to address this issue, a binary classifier to distinguish resting moments from periods of activity is presented. This classifier is trained with handcrafted pose descriptors and then it should be able to precisely spot starting and ending points of each gesture. Later, the prediction outputted by the classification module can be adjusted to the boundaries produced by this trained binary classifier.

3.1. Moving Pose Descriptor

Central to this methodology is the moving pose descriptor aforementioned that includes 7 subsets of logical features. The descriptor is calculated based on 11 upper body joints – not considering the unstable wrist joints – depicted in Figure 3.1 (a). Their raw, i.e. pre-normalization, positions in a 3-dimensional coordinate system associated with the Kinect sensor are denoted as $\boldsymbol{p}_{\mathrm{raw}}^{(i)} = \{x^{(i)}, y^{(i)}, z^{(i)}\}$ with $i \in \{0, \dots, 10\}$ and $i = 0$ corresponding to the *HipCenter* joint.

A good descriptor should capture not only static pose information, but also body joints' kinematics at a given moment in time. Accordingly, this module follows the procedure proposed by (Zanfir, Leordeanu and Sminchisescu, 2013) and thereby it firstly calculates normalized joints' positions, as well as their velocities and accelerations within a short time window around the current frame.

The skeleton is represented as a tree structure with the *HipCenter* joint playing the role of a root node. Its coordinates are subtracted from the rest of the vectors $\boldsymbol{p}_{\mathrm{raw}}$ in order to eliminate the influence of spatial body position. Furthermore, human subjects have differences in body and limbs size, which are not relevant for the action performed. To compensate this anthropometric discrepancies, it starts from the top of the tree and iteratively normalizes each skeleton segment, which is defined by any 2 linked joints and it corresponds to an average "bone" length. Obviously, the evaluation uses all available training data. This

data processing allows correcting absolute joints' positions, while corresponding orientations remain unchanged:

$$\boldsymbol{p}^{(i)}(t) = \boldsymbol{p}_{\text{raw}}^{(i-1)}(t) + \frac{\boldsymbol{p}_{\text{raw}}^{(i)}(t) - \boldsymbol{p}_{\text{raw}}^{(i-1)}(t)}{\left\|\boldsymbol{p}_{\text{raw}}^{(i)}(t) - \boldsymbol{p}_{\text{raw}}^{(i-1)}(t)\right\|} b^{(i-1,i)} - \boldsymbol{p}_{\text{raw}}^{(0)}(t), \tag{3.1}$$

where $\boldsymbol{p}_{\text{raw}}^{(i)}$ is the current joint, $\boldsymbol{p}_{\text{raw}}^{(i-1)}$ is its direct predecessor in the tree and $b^{(i-1,i)}$ is the estimated average "bone" length that links both joints. It is important to emphasize that, in this moment, $i \in \{1, \dots, 10\}$ because the *HipCenter* joint is localized at the origin of the actual coordinate system. Finally, the output $\boldsymbol{p}$ contains the normalized joints' positions. Once the calculus is completed, a Gaussian smoothing is performed along the temporal dimension to decrease the impact of skeleton volatilities. This operation is accomplished sliding a window with 5 frames of size and a unitary standard deviation.

Velocities and accelerations of each joint are calculated as the first and second derivatives, respectively, of its normalized positions, as elucidated by the following equations:

$$\delta\boldsymbol{p}^{(i)}(t) \approx \boldsymbol{p}^{(i)}(t+1) - \boldsymbol{p}^{(i)}(t-1), \tag{3.2}$$

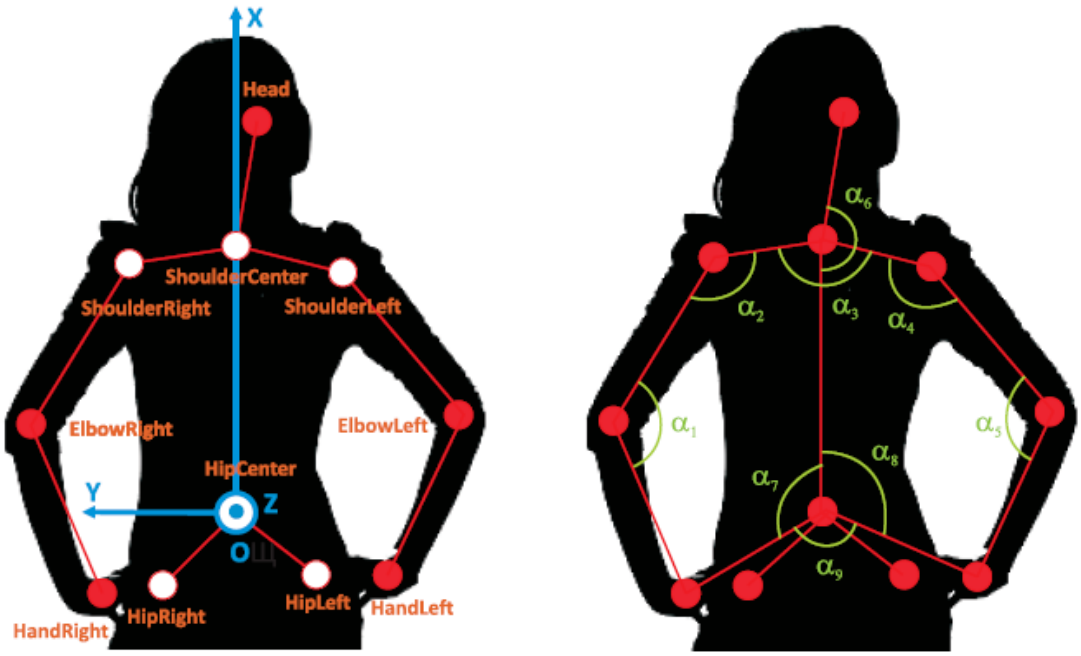
$$\delta^2\boldsymbol{p}^{(i)}(t) \approx \boldsymbol{p}^{(i)}(t+2) + \boldsymbol{p}^{(i)}(t-2) - 2\boldsymbol{p}^{(i)}(t). \tag{3.3}$$

The classification module will also be implemented based on this moving pose descriptor. Therefore, it is even more important to embed the velocity and acceleration of each joint because these features capture relevant information about different actions. For example, changes in direction as well as in speed produce nonzero acceleration, which is useful to separate gestures involving circular motions from others comprising linear actions.

In order to get a more accurate moving pose descriptor, it can be augmented with a set of characteristic angles and pairwise distances, as described by (Neverova *et al.*, 2015a). Starting with inclination angles, they are formed by all triples of anatomically connected joints (i, j, k). As portrayed in Figure 3.1 (b), 2 virtual "bones" between *HipCenter* and each of the hands were added in such way that 9 inclination angles, $\alpha^{(i,j,k)}$, can be considered:

$$\alpha^{(i,j,k)} = \arccos \frac{(\boldsymbol{p}^{(k)} - \boldsymbol{p}^{(j)}) \cdot (\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)})}{\|(\boldsymbol{p}^{(k)} - \boldsymbol{p}^{(j)})\| \, \|(\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)})\|}. \tag{3.4}$$

To demystify any doubt, the mathematical operation denoted by "." symbolizes, as in most of the scientific documents, the dot product between two arrays.



(a) (b)

Figure 3.1. The pose descriptor is based on 11 upper body joints relevant to the task: (a) representation of all the relevant joints and the "bones" that link each pair of them; (b) inclination angles formed by all triples of connected joints if considering 2 virtual "bones" (Neverova *et al.*, 2015b).

Azimuth angles provide additional information about the pose in the coordinate system associated with the body. To compute them, it is necessary to previously use PCA on the 6 torso's joints – *HipCenter*, *HipLeft*, *HipRight*, *ShoulderCenter*, *ShoulderLeft*, *ShoulderRight* – to obtain 3 vectors forming the basis: $\{\boldsymbol{u}_x, \boldsymbol{u}_y, \boldsymbol{u}_z\}$, where $\boldsymbol{u}_x$ is aligned with the spine, $\boldsymbol{u}_y$ is approximately parallel to the shoulder line and $\boldsymbol{u}_z$ is perpendicular to the torso. Then for each pair of connected "bones", $\beta^{(i,j,k)}$ is an angle between the projections of the second "bone", $\boldsymbol{v}_2$, and the vector $\boldsymbol{u}_x$, $\boldsymbol{v}_1$, on the plane perpendicular to the orientation of the first "bone". As in the case of inclination angles, 2 virtual "bones" were included:

$$\boldsymbol{v}_1 = \boldsymbol{u}_x - \left(\boldsymbol{p}^{(j)} - \boldsymbol{p}^{(i)}\right)\frac{\boldsymbol{u}_x \cdot \left(\boldsymbol{p}^{(j)} - \boldsymbol{p}^{(i)}\right)}{\left\|\left(\boldsymbol{p}^{(j)} - \boldsymbol{p}^{(i)}\right)\right\|^2},$$

$$\boldsymbol{v}_2 = \left(\boldsymbol{p}^{(k)} - \boldsymbol{p}^{(j)}\right) - \left(\boldsymbol{p}^{(j)} - \boldsymbol{p}^{(i)}\right)\frac{\left(\boldsymbol{p}^{(k)} - \boldsymbol{p}^{(j)}\right) \cdot \left(\boldsymbol{p}^{(j)} - \boldsymbol{p}^{(i)}\right)}{\left\|\left(\boldsymbol{p}^{(j)} - \boldsymbol{p}^{(i)}\right)\right\|^2}, \tag{3.5}$$

$$\beta^{(i,j,k)} = arccos\frac{\boldsymbol{v}_1 \cdot \boldsymbol{v}_2}{\|\boldsymbol{v}_1\| \, \|\boldsymbol{v}_2\|}.$$

Bending angles, $\gamma^{(i)}$, are a set of angles between the basis vector $\boldsymbol{u}_z$, perpendicular to the torso, and the normalized joints' positions. Note that there is not logic in calculating $\gamma^{(0)}$ because its computation would lead to a division by zero:

$$\gamma^{(i)} = \arccos \frac{\boldsymbol{u}_z \cdot \boldsymbol{p}^{(i)}}{\|\boldsymbol{p}^{(i)}\|}. \tag{3.6}$$

The pairwise distances between all the human body's joints, $\rho^{(i,j)}$, represent the last feature that composes this moving pose descriptor. They can be computed as described below:

$$\rho^{(i,j)} = \left\|\boldsymbol{p}^{(i)} - \boldsymbol{p}^{(j)}\right\|. \tag{3.7}$$

Combining all features, a 182-dimensional pose descriptor for each video frame is achieved. To reiterate, there are 33 normalized joints' positions, 33 velocities, 33 accelerations, 9 inclination angles, 9 azimuth angles, 10 bending angles and 55 pairwise distances. Once obtained the moving pose descriptor for each frame of all video sequences, all values, feature by feature, are normalized to zero mean and unit variance.

3.2. Segmentation Architecture

This section describes how the ANN is defined, namely what type of data it needs to be trained, their hyperparameters and the output that it is able to provide. Recall that an ANN takes some inputs in the first layer's neurons and each neuron is connected to every node in the following layer. These connections have an associated weight and bias. Then, an activation function is applied to the sum of the values obtained from each connection. The output layer is connected to the layer before by the same type of connections, but it has not an activation function; instead the final scores are computed by an output function. This establishes a non-linear relationship between inputs and outputs. A Multi-Layer Perceptron (MLP) is a feedforward ANN that is commonly used for binary classification tasks. Accordingly, this particular case will be implemented in this stage.

The MLP architecture chosen to distinguish resting moments from periods of activity is presented in Figure 3.2. Since the input for this network is the moving pose descriptor described above, the input layer has 182 neurons. Remember that this is the number of features comprised by the descriptor. There are 2 hidden layers with 100 neurons each. Furthermore, different activation functions are being used, since the first layer applies

a Rectified Linear Unit, while the second one employs the hyperbolic tangent. The Table 3.1 shows that using 2 hidden layers instead of 1 provides slightly better results. The output layer has a single neuron and the output function is the sigmoid. This layer could have 2 neurons, which is the number of classes, but once dealing with a binary classifier, it can be trained to output 1 when there is motion and zero otherwise. The chosen network was optimized with Scaled Conjugate Gradient (SCG) and it took 494 iterations and 64 minutes to train.

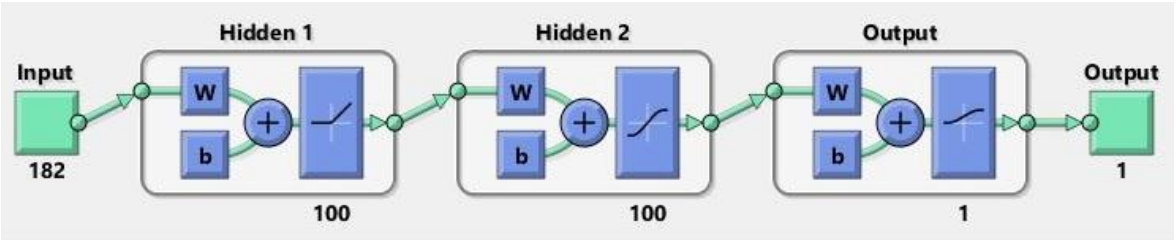


Figure 3.2. MLP architecture used for the segmentation module. The input layer has 182 features, the 2 hidden layers have 100 neurons each and the output layer has only 1 neuron. The first and second layers' activation functions are the ReLU and hyperbolic tangent, respectively, while the output layer uses the sigmoid function.

Table 3.1. Several designs were tried to the MLP of the segmentation module. It is possible to confer the best results are achieved with 2 hidden layers and using a ReLU activation function in the first hidden layer. Increasing the number of neurons in the first hidden layer does not appear to provide better results.

	Hidden Layers' Size	Activation Functions	Accuracy (%)
1	[100]	Hyperbolic Tangent	94.3
2	[100]	ReLU	93.9
3	[300]	ReLU	93.8
4	[100 100]	Hyperbolic Tangent – Hyperbolic Tangent	94.3
5	[100 100]	ReLU – Hyperbolic Tangent	95.2
6	[300 100]	ReLU – Hyperbolic Tangent	94.6

To train the network, all training frames labelled with some gesture class are used as positive examples, while a set of frames right before and after of such gesture are considered as negatives. This data set is randomly split into training, validation and testing sets. The detailed strategy allows to assign to each frame a label of "motion" or "no motion" with an overall accuracy of 95.2%.

3.3. Results and Discussion

After all the weights and biases had been learned, the Multi-Layer Perceptron is ready to be used. The moving pose descriptors that define each video are sequentially evaluated by the network and the output is a score for each frame suggesting if it contains motion. Once the prediction is noisy, it must be smoothed; local regression is applied using weighted linear least squares and a second degree polynomial function. This regression uses a window that covers 2.5% of the video's length. In order to effectively complete the binary classification, a threshold is used, imposing that frames with a score higher than 0.4 must be attached with the "motion" label. Otherwise, the frame is classified with the "no motion" label. Sometimes, the output function, even the smoothed version, has peaks that do not mean activity and, if they remain unchanged, they will lead to some wrong classifications. Therefore, the last procedure enforces each gesture to last at least 12 frames. The result of this technique can be visualized in Figure 3.3.

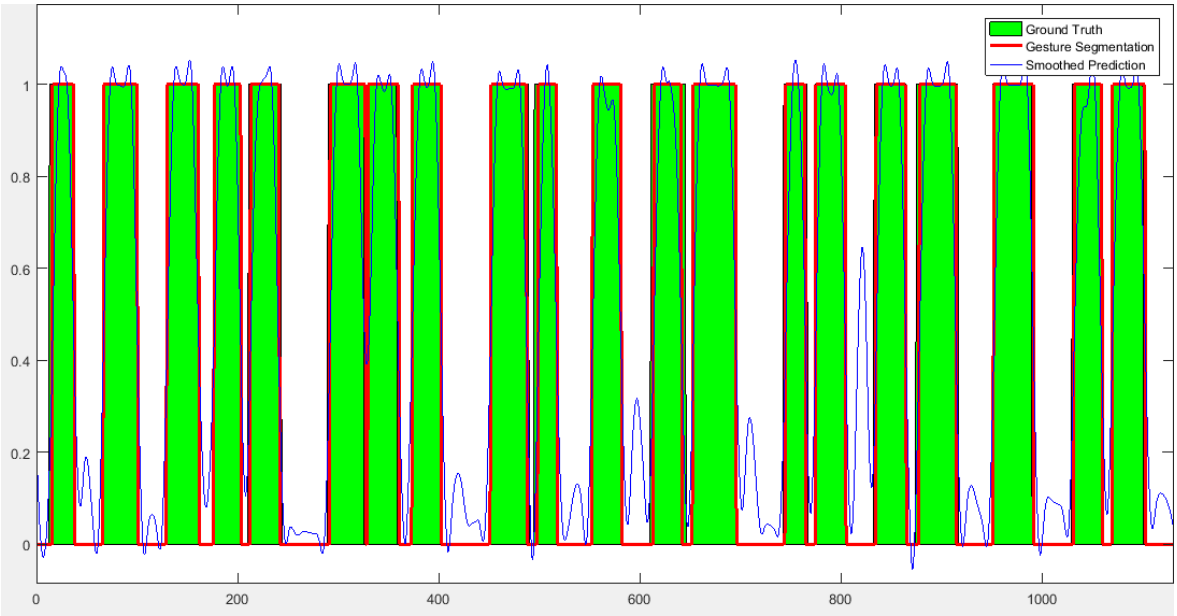


Figure 3.3. Since each green bar represents the ground truth of a single gesture, it can be seen that the constructed model is able to detect and localize all the 20 gestures contained in the sample #727. This sample belongs to the test set. The time is represented in the horizontal axis.

As shown above, when the subject performs gestures separated by some time interval and assumes a static pose between them, the model works very well. This occurs most of the times. However, some subjects execute consecutive gestures – in some cases, with no one frame between them – and others assume a very dynamic, unquiet pose over all video sequence. In these circumstances, the segmentation task is quite challenging and the output looks like the Figure 3.4. In order to make the model more accurate and soften the problem, a complementary approach is tested. Primarily, the mean length of each gesture is studied, since the targets of this upgrade are only the segments widely enough to cover more than a single gesture. Once defined the threshold, all frames of segments whose length is higher than that value are evaluated on a second ANN, which is different from the previous one because it was trained based only on the velocities. The aim of this try is to detect "no motion" moments even when the subject does not lower their arms between 2 gestures. In other words, it can be said that the normalized joints' positions and the other features lose importance in favour of the velocity and, hence, this is a velocity-based segmentation. As it can be seen in Figure 3.4, the results were not what were expected, since the segmentation is frequently illogic. Notice that some partitions resulting from the separation have a small, almost meaningless length. For this reason, from now on, the velocity-based segmentation is abandoned and the problem will be addressed during the classification section.

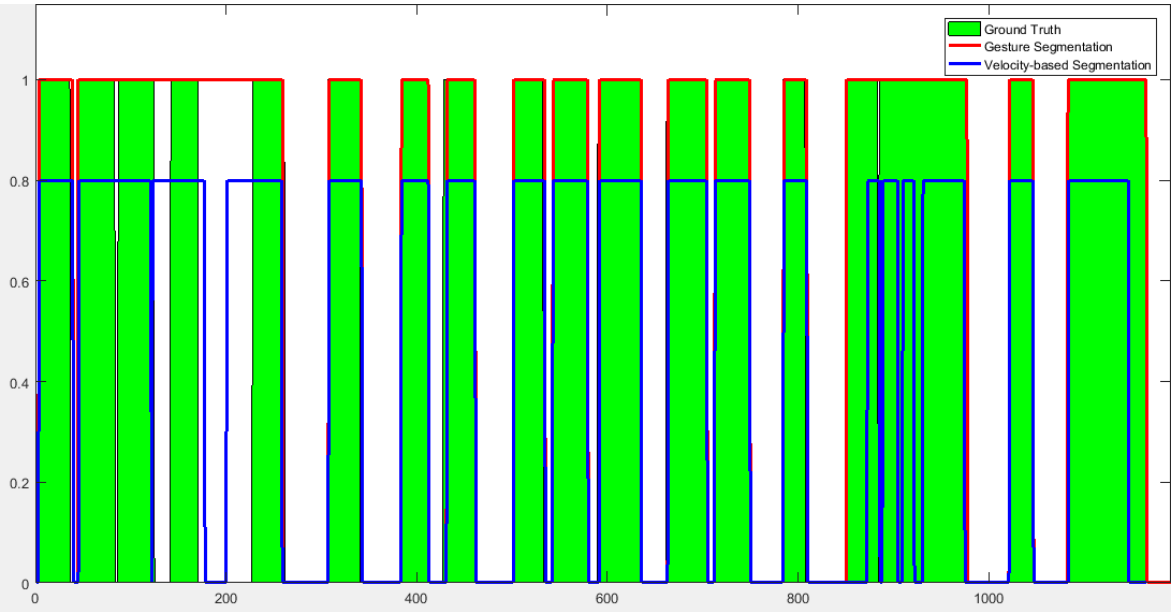


Figure 3.4. To appreciate the work done by velocity-based segmentation, the attention must be focused on the widest segments of the sample #721. In the first very large portion, the segmentation is not as good as the primary segmentation for short gestures. On the other hand, in the second wider portion, very small segments were considered, which is obviously a mistake. The output of the velocity-based segmentation is lower than the unity only to improve the visualization. The time is represented in the horizontal axis.

There is another special situation that deserves to be emphasized. When the subject performs a distractor gesture – out of the 20 categories – the model works normally and segments it, not knowing that it is dealing with an unlabelled gesture. The situation illustrated in the top of the Figure 3.5 cannot be solved during the segmentation because there is a gesture being effectively performed. Hence, the unlabelled gestures will be another task for the classification model, which will have to be accurate enough to predict that those sequences do not belong to the vocabulary. Yet for the video sequence addressed in Figure 3.5, the evolution exhibited by the features over time is shown. Since the data were normalized to zero mean and unit variance, the adopted scale, which is visible in the colour bar, allows that 99.73% of all values remain unchanged. In other words, all values above 3 were considered equal to 3 and the same procedure was employed in the negative values.

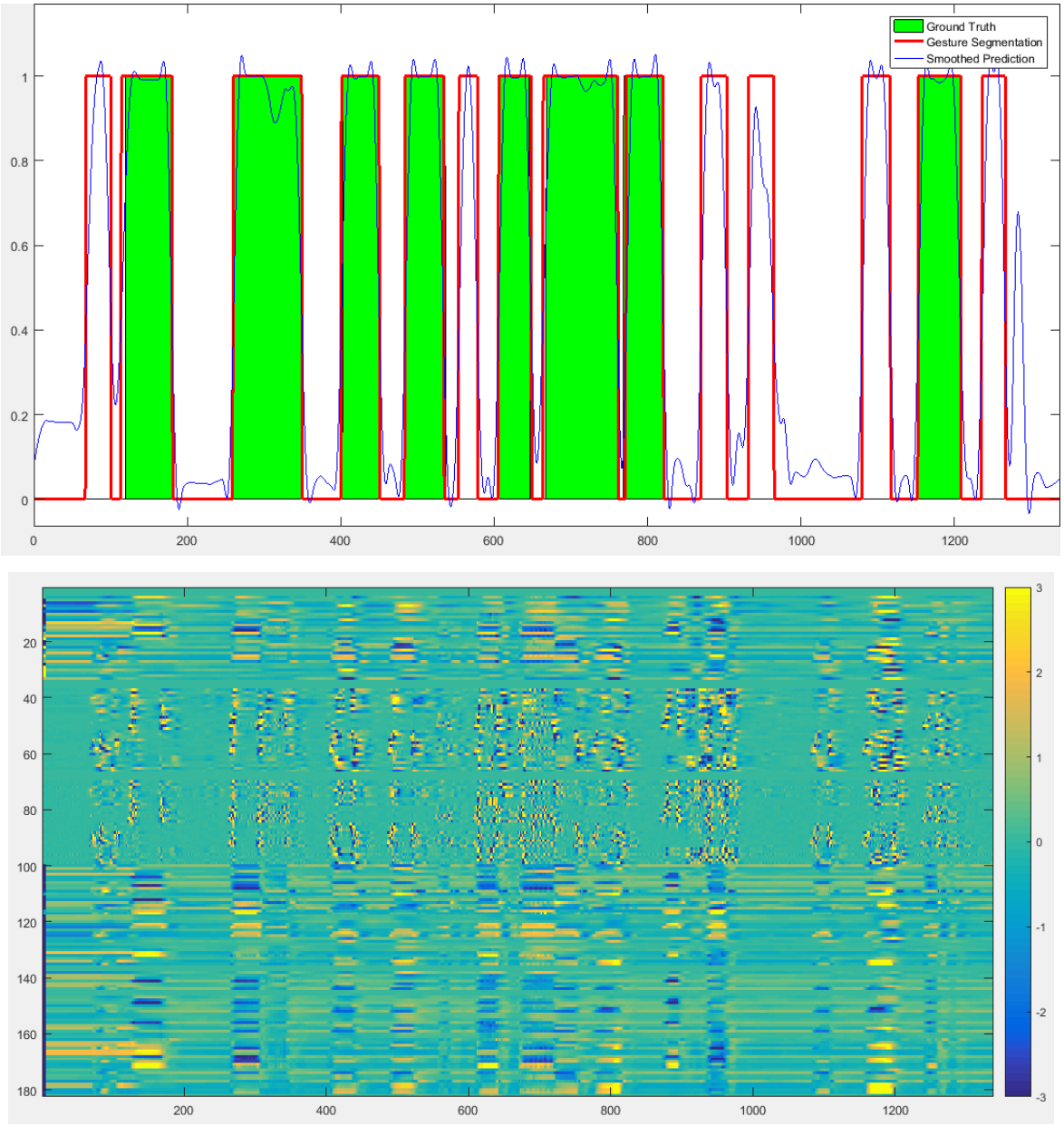


Figure 3.5. Data from the sample #703. On the top, the time is represented in the horizontal axis and it is possible to realize that this sample contains several unlabelled gestures. The segmentation model worked well and delimited all gestures. Below, the features over time are colour mapped. The horizontal axis holds the temporal evolution, while the vertical axis has the number of features.

4. GESTURE CLASSIFICATION

In the previous chapter, the task of gesture detection and segmentation was addressed. To recall, an ANN capable to distinguish resting moments from periods of activity was developed. After the training, for all the test video sequences, the ANN was used to do framewise cataloguing with a label of "motion" or "no motion". Since it is now known the moment when each gesture is performed, it is necessary to classify it. Accordingly, in this chapter, diverse approaches will be built up to complete the task. It must be remembered that this chapter has to handle with 2 main difficulties coming from the segmentation task: the subjects that execute consecutive gestures and the unlabelled, distractor gestures.

4.1. Method 1

When the aim of work is to classify static gestures, framewise classification is a reasonable strategy. Such type of work copes with very distinctive gestures because, when looking at a single frame, it must be possible to recognize what gesture is being performed. On the other hand, the current dissertation intends to label dynamic gestures and hence framewise classification is not an option. In this sense, it was decided to base all methods on a sliding window, but the details will be explained later on.

The moving pose descriptor constructed in the previous chapter will be essential here too. Recall that a set of 182 handcrafted features was developed in order to characterize each frame of all video sequences. Each descriptor includes normalized joints' positions, velocities, accelerations, inclination angles, azimuth angles, bending angles and pairwise distances.

4.1.1. Collection of Data

The application of a sliding window can be an acceptable approach because it captures spatial information along the time and uses a scale defined by its hyperparameters. Therefore, one of the first decisions that has to be made is the number of descriptors included in each dynamic pose – this last term means a sequence of pose descriptors sampled at a

given temporal step, s_{DP}, and concatenated to form a spatio-temporal vector. Hereafter, the number of descriptors will be called as sequence length, l_{DP}, since it is dealing with a sequence of a few chained descriptors. Another variable was already introduced and it is the step between the selected pose descriptors. Notice that these parameters are sufficient to define the total extension of the sliding window. As depicted in Figure 4.1, if the sequence length is 3 and the temporal step is 4, then the sliding window will have the ability to collect information from a temporal span that covers 9 frames.

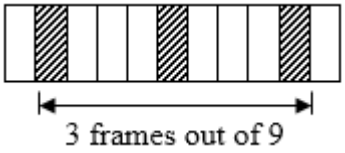


Figure 4.1. Schematic representation of a sliding window. In this particular case, it allows to construct a dynamic pose with 3 pose descriptors, which are sampled with a temporal step of 4 frames.

The sliding window's length was defined above, but most of gesture performances, or even all, last more than 9 frames. Hence, it is crucial to define where to apply the first window and until where it must be slid. It was decided that, for each labelled gesture, the first window is applied in such manner that its first frame is coincident with the initial frame of the gesture. Then, the window is slid until it is possible, i.e. until the moment an extra iteration will put the window collecting information that does not belong to the gesture. To complete this part, it is necessary to define another temporal step, s_{W}, which will command the progress of the window after each iteration. In order to give continuity to the example initiated above, it will be assumed that s_{W} is equal to 3. So, if a gesture is performed during 18 frames, the sliding window will be applied in 4 different locations and the same number of dynamic poses will be constructed. Next, these 4 sequences of pose descriptors and the respective target class will be in the inputs to train an ANN, which must be capable to classify each dynamic pose into a gesture class.

Assuming that a gesture is performed during a time interval of 14 frames, the hyperparameters defined above allow the application of only 2 windows. In other cases, the small extension of a gesture could allow that only 1 window, or even none at all, is applied. So, to guarantee that a minimum quantity of data about each gesture is gathered, another constraint is defined, which is the minimum number of windows whose application is

required, min_{W}. The fusion of the 4 hyperparameters defined here conduct to a minimum gesture's length, min_{GL}:

$$min_{\mathrm{GL}} = (l_{\mathrm{DP}} - 1) \times s_{\mathrm{DP}} + 1 + (min_{\mathrm{W}} - 1) \times s_{\mathrm{W}}. \quad (4.1)$$

When the length of a gesture is lower than the minimum value, interpolation is used to resize the data. In this way, there is always a minimum amount of information to slide min_{W} windows and the rest of the process is the same than before.

4.1.2. Classification Architecture

Proceeding as described above for all labelled gestures of each training sequence, a significant amount of data is collected, which is now used to train an ANN. The architecture chosen is presented in Figure 4.2. The input for this neural network is a dynamic pose comprising 3 descriptors and, hence, the input layer has 546 neurons. There are 2 hidden layers, being the first one composed by 300 neurons and the second one by only 100 neurons. Both hidden layers use hyperbolic tangent as the activation function. The output layer has 20 nodes, which is the number of classes, and the output function is the Softmax. The network was trained with SCG and the process took 658 iterations and almost 67 minutes to be completed.

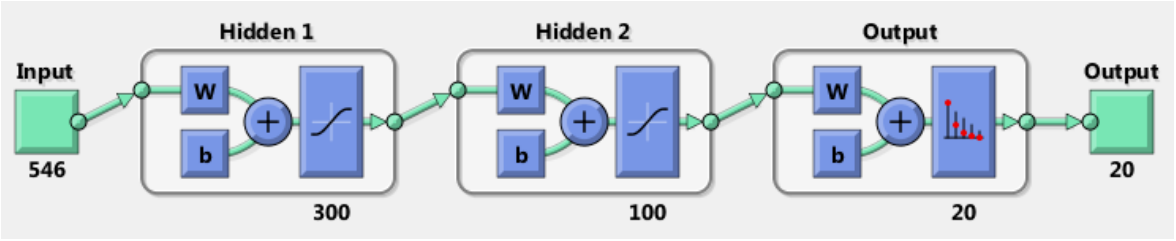


Figure 4.2. Network's architecture used to do the classification. The input layer has 546 features, the 2 hidden layers have 300 and 100 neurons, respectively, and the output layer has 20 neurons. The first and second layers' activation function is the hyperbolic tangent, while the output layer uses the Softmax.

The architecture of this ANN was optimized in a similar way to the one used in the segmentation task. Choosing this structure and the hyperparameters shown in Table 4.1, each gesture was classified with an overall accuracy of 90.2%.

4.1.3. Classification Process

In this moment, there is a trained classifier able to reasonably allocate each dynamic pose of 3 descriptors into the correct class and there is also information about the moments when the gestures are performed. So, it is necessary to construct dynamic poses – as described in the subsection Collection of Data – to characterize these moments and pass them through the network.

Even before the classification, the mean length of all gestures in the training set is studied. This quick estimation helps to predict if a segmented portion includes more than one gesture. Thus, based on the results, a value for the segment's length, lim_{SL}, is defined, above which it is assumed that it contains 2 or more gestures and, as described later on, a different strategy is employed.

When the segment's length is lower than the boundary defined above, it is assumed that the segment contains only 1 gesture and the window is slid across all of its frames. For each window, the dynamic pose is constructed and evaluated in the network, which outputs a vector of classes' scores. If the score of the winning class is higher than a certain threshold, $threshold_1$, this gesture class is saved. At the end, if the number of savings of the most common class is higher than half the number of the windows applied, all the segment under evaluation is classified with this gesture class. Otherwise, the segment remains without classification. Note that the threshold and the minimum number of saves of the most common class is the strategy adopted to deal with the unlabelled gestures.

On the other hand, when the segment's length is equal or higher than lim_{SL}, it is expected that the segment surrounds more than 1 gesture and the intention is to detect and classify all of them. Again, for each window, some descriptors are concatenated to form a dynamic pose, which is evaluated in the network. If the score of the winning class is higher than a threshold – $threshold_2$, different from the one defined in the previous paragraph – this gesture class is saved. When the same gesture class is identified in a minimum number of consecutive windows, $cons_{\mathrm{W}}$, it is assumed that this gesture was effectively performed. Then, all the frames comprised by the consecutive windows classified with the same gesture class are assigned to this class. After that, the process continues and the next window starts immediately upon the end of the gesture detected. As it will be seen further on, this strategy seems to work reasonably.

As it is possible to realise along the previous description, there are 8 capital hyperparameters defining the structure of the sliding window, the process of collecting data and the procedure of classification. Over 70 different combinations of these hyperparameters were tested and the one that conducted to the best result is shown in Table 4.1.

Table 4.1. The values of all the 8 hyperparameters that conducted to the best result. The first 4 hyperparameters are fundamental to define the sliding window's layout and its application. Hence, its modification force to train a new network. The last 4 hyperparameters delineate the classification process.

Training				Classification			
l_{DP}	s_{DP}	s_{W}	min_{W}	lim_{SL}	$threshold_1$	$threshold_2$	$cons_{\mathrm{W}}$
3	4	2	5	56	0.8717	0.6255	3

4.2. Method 2

Using wider sliding windows leads to noisy predictions of gestures, particularly on their initial and final phases, and in some cases it conducts to overlapping of gesture performances. On the other hand, dynamic poses that are too short are not sufficiently discriminative, as most of gesture classes at their initial or final stages have a similar appearance. The key point is to find out what the values of sequence length and step between the sampled descriptors that lead to a higher precision. Once this is a difficult task, another approach is to construct dynamic poses with descriptors sampled at different temporal steps, as it was done by (Neverova *et al.*, 2015a). Notice that varying the value of step allows the model to leverage multiple temporal scales for prediction, thereby accommodating differences in tempos and styles of articulation of different users.

In agreement with what was said in the paragraph above, this second method is very similar to the first one. The improvement achieved is precisely the introduction of multiple temporal scales, since it uses 3 windows with different steps between the selected descriptors.

For each window, the process of collecting data and training the network is exactly equal to the one described in Method 1. The hyperparameters that discriminate the sliding windows are the same as those shown in Table 4.1, except the step between the selected descriptors, s_{DP}. Concretely, the first window has a step equal to 4, the second one

uses a step of 3 and last one is constructed with a step of 2. Regarding to the hyperparameters delineating the classification procedure, $threshold_1$ and $threshold_2$ are substituted by 0.6014 and 0.6033, respectively. The 3 networks have a similar architecture to the one depicted in Figure 4.2. It is important to say that the train of the 3 networks took almost 4 hours, in which 2184 iterations were completed.

After the training and during the classification, each window slides across the entire segmented portion and the scores are combined with weights: 0.4895, 0.4576 and 0.0529, respectively. Once obtained a single vector of classes' scores, the classification follows the same conditions as defined before. Note that these thresholds and weights were optimized with random search and more than 650 different combinations were tested. Looking to the weights used, it is possible to realise that the discriminative power of dynamic poses depends on the sampling step, s_{DP}, and achieves a maximum value in the case of large sliding windows. The importance of the thinner window in the method presented is very low, 5.29%.

4.3. Method 3

The third and last method discussed in this dissertation has once more a decision structure identical to the first one, but at the same time there is a big difference and some years of research between both methods. Actually, this method discards all the 182 handcrafted features characterizing each frame and it introduces deep learning, in particular the architectures widely studied in chapter CONVOLUTIONAL NEURAL NETWORKS, which have the capacity to learn the most important features and representations by itself.

In order to collect data to train the ANN, the first method gathered a fixed number of pose descriptors to construct a dynamic pose for each window applied. The algorithm now introduced uses the same sliding window, but instead of employing the pose descriptors, it collects the RGB images for the same frames. In this sense, the sequence length is now related to the number of sampled RGB images for each window. After, the segmentation mask is used to find out the position of the subject and the RGB images are cropped. As explained previously, only the upper body is relevant to distinguish gestures and, for this reason, the lower body is also removed. This procedure allows to extract noise from the image. Finally, a new image is constructed, where each pixel value is a mean of the

corresponding pixel values in the chosen RGB images. This global transformation can be observed in Figure 4.3. Proceeding in the same way for all labelled gestures covered by the training set, they are collected almost 110,000 modified images, which will be the input to train the CNN.

Figure 4.3. Representation of the process that conducts to the image capable to contain data from a temporal span. The first 3 images are frames sampled from the video sequence, which were cropped to remove noise. The image on the right is like the overlap of the previous 3.

The architecture developed for the CNN and the details of learning are introduced in the next sections. Once the network is trained, the classification can be made. This task uses the same structure and the hyperparameters defined in Method 1. Again, the unique difference is the set of data used as input of the network to achieve the classes' scores.

4.3.1. CNN Structure

The complete structure of the CNN is depicted in Figure 4.4. The spatial dimensions of the input images are 224 × 224 and they have 3 channels, so that the input size is exactly 224 × 224 × 3. The first convolution layer consists of 96 feature maps produced by 7 × 7 filters, which are applied with a stride of 2 in both spatial dimensions. Afterwards, the activations are corrected with the ReLU activation function and a max pooling with size 3 × 3 and stride 2 is applied. The second convolutional layer uses 256 feature maps obtained with 5 × 5 filters, followed by the ReLU activation function and the same max pooling operation. The third convolutional layer is composed of 384 feature maps achieved with 3 × 3 filters, which are employed with a stride 1, followed only by the ReLU activation function. After all the convolutional layers, 2 fully connected layers composed by 4096 and 1024 neurons, respectively, are applied. Finally, the output layer has 20 neurons and its activation function is the Softmax. This architecture has its basis in the network constructed by (Zeiler and Fergus, 2014).

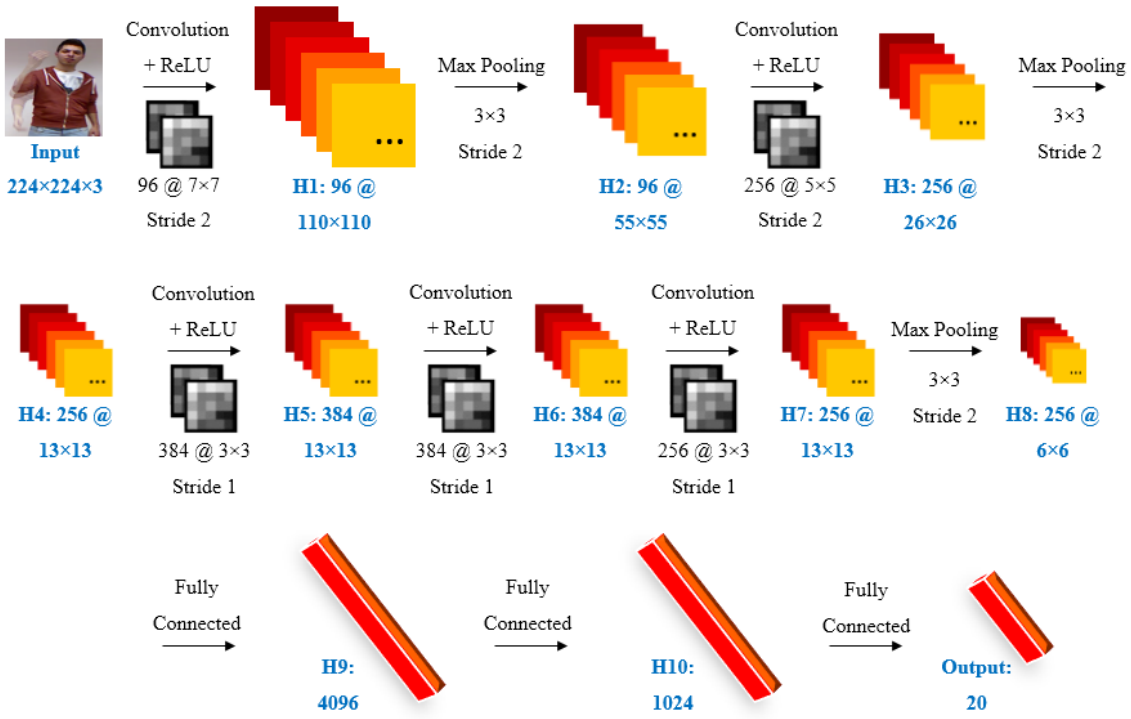


Figure 4.4. CNN architecture. The input is a 224 × 224 × 3 image. This is convolved with 96 different filters, each of size 7 × 7, using a stride of 2 in both spatial dimensions. The resulting feature maps are repaired with the ReLU and they are pooled using the max operation with size 3 × 3 and a stride of 2. Similar operations are repeated in next layers. The last 2 hidden layers are fully connected and the output layer uses the Softmax classifier. All filters and feature maps are square in shape.

4.3.2. Details of Learning

The updates of the network's parameters are performed using Gradient Descent with Momentum and a fixed momentum coefficient of 0.9. In order to achieve faster convergence and also to prevent overfitting, batches of 128 images are used. The weights of the CNN are randomly initialized from a normal distribution with zero mean and standard deviation 0.01. Furthermore, the initial bias is always zero. The learning rate is initialized at 0.01 and it is divided by a factor of 10 after each group of 8 epochs. The learning process is interrupted when there are 20 completed epochs.

4.4. Results and Discussion

4.4.1. Overall Results

The top 10 scores of the *2014 Looking At People Challenge*'s third track are reported in Table 4.2. These results will be the basis of comparison to those here obtained, which are shown in Table 4.3.

Table 4.2. The top 10 results of the *2014 Looking At People Challenge*'s third track. In total, there were 17 different submissions.

Rank	Team	Score	Rank	Team	Score
1	(Neverova *et al.*, 2015a)	0.8500	6	(Wu, 2015)	0.7873
2	(Monnier, German and Ost, 2015)	0.8339	7	(Camgöz, Kindiroglu and Akarun, 2015)	0.7466
3	(Chang, 2015)	0.8268	8	(Evangelidis, Singh and Horaud, 2015)	0.7454
4	(Peng *et al.*, 2015)	0.7919	9	(Nandakumar *et al.*, 2013)	0.6888
5	(Pigou *et al.*, 2015)	0.7888	10	(Chen *et al.*, 2015)	0.6490

Table 4.3. Performance of the different methods described in the text above, which are evaluated with the aforementioned Jaccard index.

Method	Score
1	0.7008
2	0.7104
3	0.0359

Although the best Jaccard index achieved by (Neverova *et al.*, 2015a) is 0.8500, notice that their best result using only skeleton data and a single sliding window is 0.7891. When they introduced different temporal scales, they improved to 0.8080. Their top Jaccard index was achieved combining different modalities of data, namely skeleton data and RGB-D images. In this dissertation, a single sliding window allows to achieve a Jaccard index of 0.7008, while the introduction of different temporal scales lead to a score of 0.7104. Once the results are known, it is possible to realize that the results obtained are almost in according with the other state of the art techniques. Notice that the difference between the Jaccard indices achieved with a single sliding window is less than 9 percentage points. On the other hand, the improvement reached with the introduction of different temporal scales is, approximately, half than the progress made by (Neverova *et al.*, 2015a) with the same upgrade. Despite everything, the top Jaccard index achieved here would be sufficient to place this dissertation in the ninth position of the challenge's ranking.

The deep learning approach was tested separately and the results are very poor, much worse than those obtained with the previous methods, with a few reasons to be pointed out. In the first place, it is reasonable to think that the images used to train the CNN, like the one depicted in Figure 4.3, are not discriminative or robust enough, in such manner that distinguishing different gestures is a hard task for the network. In retrospective, recall also the process of collecting data: for the training set, a window slides from the start to the end of each labelled gesture, sampling some RGB images to construct a dynamic pose. Since the action performed in the initial or final phase of a gesture – for example, raise or low an arm – is similar for many gestures, it is possible that the CNN is receiving some noisy, confusing data. In this sense, it is also possible that the number of representative images of the different gesture classes is low. Given more training data, it is expected that deep learning models will be able to become better suited for gesture classification, independent of the user. Furthermore, as this method relies on the raw data and is learned only from the training set, it suffers from some overfitting. Notice that the training accuracy is very high – almost 95.0% – and the Jaccard index is very low. In order to solve this issue, some dropout layers were introduced. However, this procedure raises the number of epochs needed to converge and, hence, the memory of the GPU (Graphics Process Unit) became a new problem, being that the GPU used is a *GeForce GTX 1050 Ti*.

4.4.2. Confusion Matrices

The confusion matrices for all the 3 methods are illustrated in Figure 4.5. They are represented in log form to emphasize the wrong predictions. Otherwise, the first 2 matrices would be almost white out of the diagonal. Look also to the Figure 4.6.

Observing the confusion matrices, it is possible to realize that some gestures are more easily recognized than others. This is the case of gesture #13 – "*Basta*" gesture – whose arms position does not resemble with the arms position of many other gestures. This is maybe the reason that helps to correctly predict the gesture. On the other hand, there are gestures that are more often well predicted with one method than with another. For example, the first method works better detecting the gesture #16, while the second method detects more accurately the gesture #9.

Notice that the gestures #14 and #15 are commonly confused between themselves. Visualizing Figure 4.6, it is perceptible that these gestures include identical arms position and motion, which lead to similar skeletal features and, hence, hamper its correct classification. However, these gestures differ in their hand pose, so that the RGB-D data and the third method can be used to reduce this confusion. To apply this solution, the third method should not consider all the upper body image, but only an image with the hand. Unsurprisingly, this is the strategy adopted by the winner of the contest.

Despite the fact that the third method is not working properly, there is an interesting conclusion that must be highlighted. All gesture classes, excluding the first one, get an incorrect classification more often than a correct one, but each gesture class has another gesture class with which it is commonly confused. This fact is perceptible by the black squares out of the diagonal. For example, the gesture #10 is frequently confused with the gesture #2, while the gesture #11 is mistaken for the gesture #3.

The last column of the confusion matrices represents the false negatives, i.e. the frames labelled with some gesture that are classified as frames belonging to the resting moments. This is one of the major problems, since there are gestures, or at least frames of these gestures, that are not being detected. On the other hand, the last line of the confusion matrices denotes the false positives, i.e. the frames belonging to the resting moments that are classified with some gesture class. One of the reasons for this phenomenon is the classification of the out of vocabulary gestures that are performed to confound.

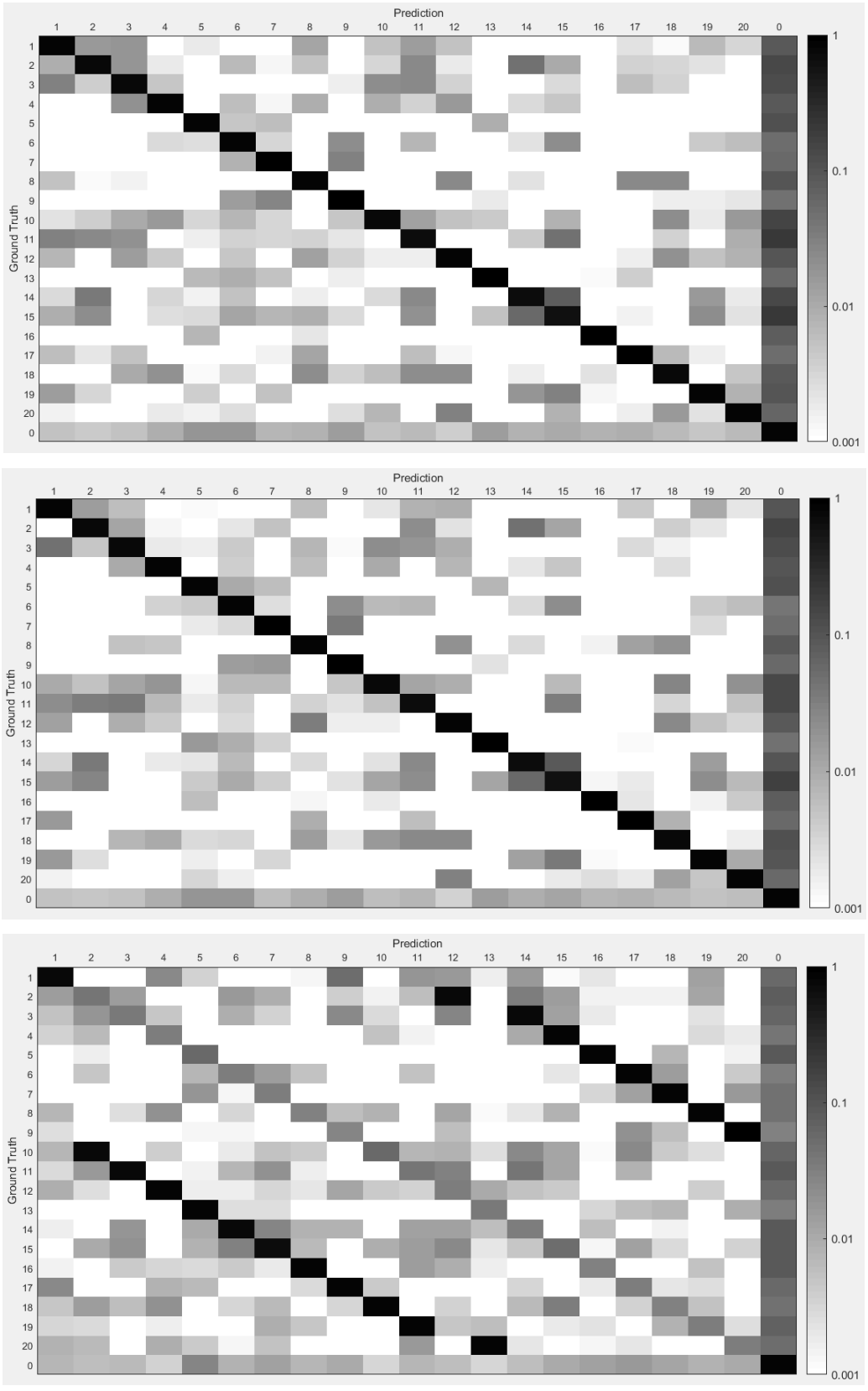


Figure 4.5. Confusion matrices, in log form, for the methods 1, 2 and 3, from the top to the bottom, respectively. The class zero corresponds to the resting moments.

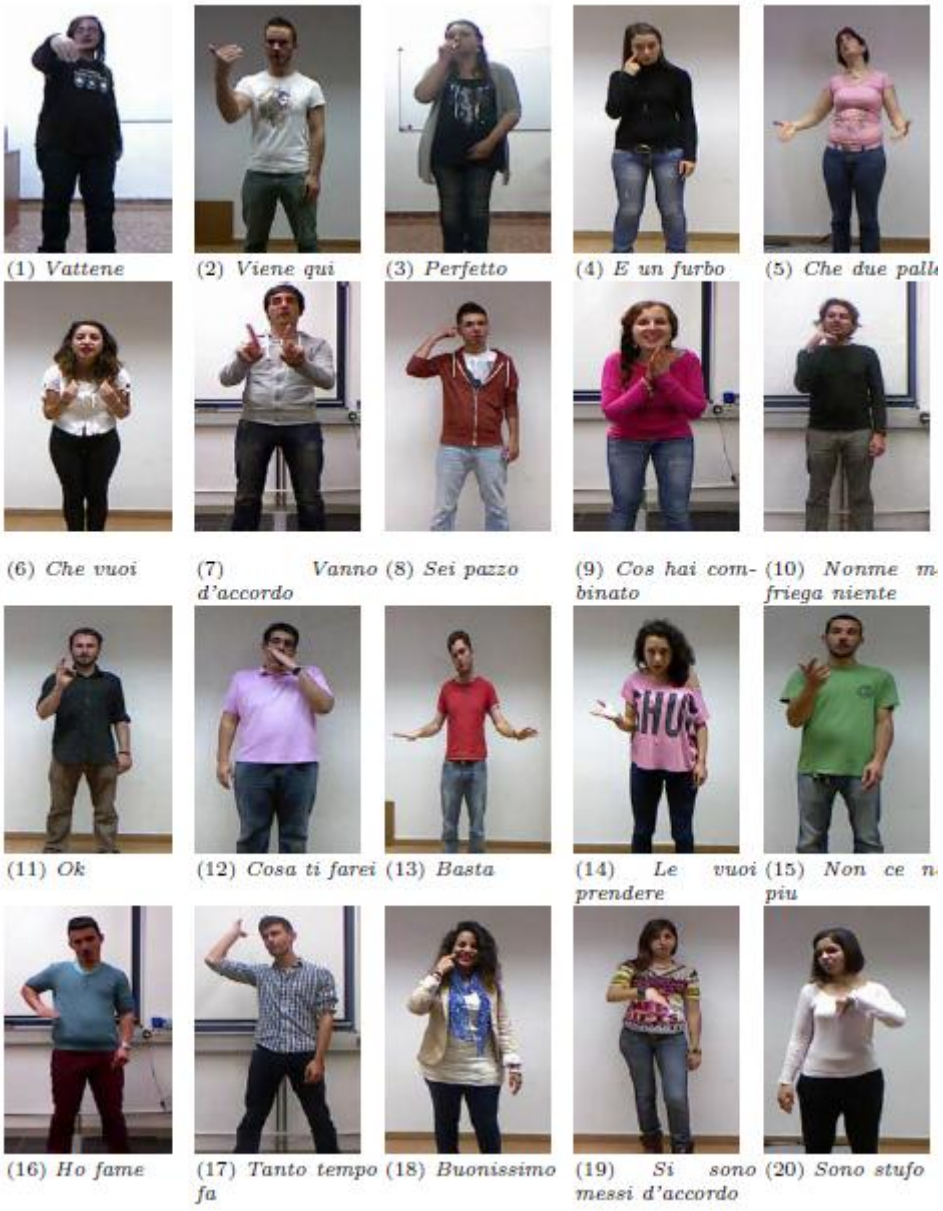


Figure 4.6. The *2014 Looking At People Challenge*'s data set includes a vocabulary of 20 Italian sign language gestures.

4.4.3. Prediction Charts

The ground truth labels and the predicted ones for the sample #703, which is included in the test set, are represented in Figure 4.7. Thus, it is possible to compare the predictions of each method for the same video sequence. Notice that each gesture class is identified with a different colour.

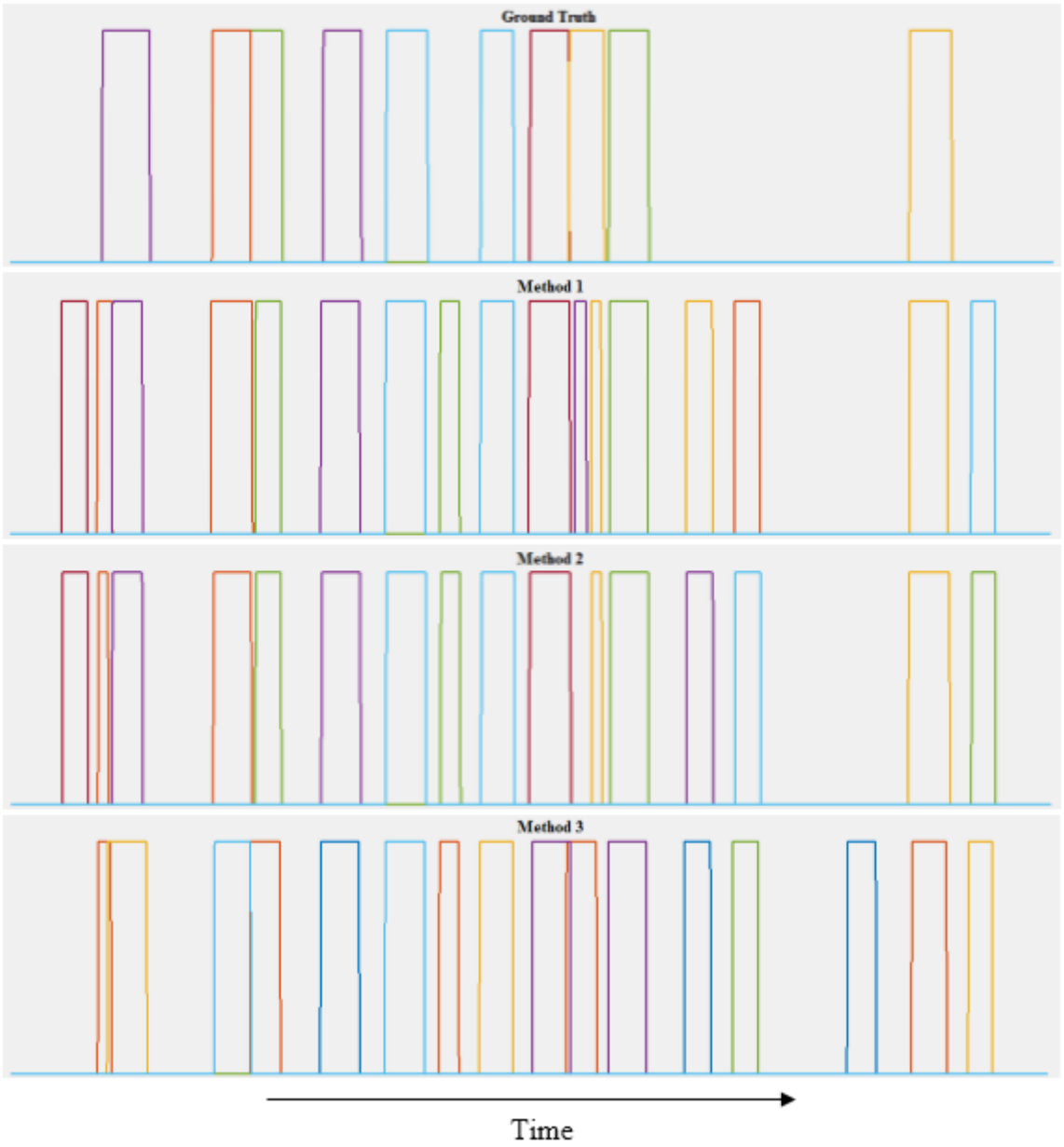


Figure 4.7. The ground truth labels and the predictions obtained with each of the methods applied.

The first image in Figure 4.7 shows that, in this sample, 10 labelled gestures are performed. The methods 1 and 2 classify all labelled gestures with an impressive precision, having only some problems in the initial phase of the first and the eighth gestures. On the other hand, the subject performs at least 5 gestures to confound, since the methods 1 and 2 make 5 predictions without ground truth label. Interestingly, note that the last 3 out of vocabulary gestures have different predictions with the methods 1 and 2. So, the junction of both methods to solve this problem is an option that deserves some attention.

As expected after the confusion matrices had been explored, the predictions obtained with the third method are unsatisfactory. Only the fifth labelled gesture is correctly classified. Relatively to the distracting gestures, this method does not classify the first one. However, there is a gesture to confound almost at the end of the sequence that is only classified with this method.

In the end, it is possible to conclude that methods 1 and 2 make reasonable predictions, while the third method is not able to give acceptable labels. On the other hand, the combination of different methods can be an interesting option to deal with out of vocabulary gestures. Figure 4.7 can be compared with the figure presented by (Wu *et al.*, 2016), in which they present the predictions obtained with their models for the same sample.

5. CONCLUSION

In this dissertation, a method to perform gesture detection and 3 different models to execute offline gesture recognition were developed. The method for gesture segmentation relied on a moving pose descriptor constructed for each frame, which results from an assembly of several handcrafted features extracted from skeleton data. Then, in order to recognize different gestures, the first and second models used sliding windows and concatenated some pose descriptors to compound dynamic poses, which were able to give the data the temporal dimension. The second method went further and used 3 different windows, which gave the model the opportunity to include various temporal scales. The last method, instead applied a CNN to automatically extract the relevant information from RGB images. Therefore, the work developed is sufficient to segment and classify some dynamic human gestures. This contrasts with considerable prior work, where the segmentation was assumed to be known *a priori*.

The present dissertation was evaluated on the *2014 Looking At People Challenge*'s data set and that allowed to gather some conclusions. First, handcrafted features built from 3D skeleton data are well suited to perform the segmentation, since they achieved an accuracy 95.2% in this task. On the other hand, these features make more mistakes during recognition and 90.2% was the maximum precision in the first model. Second, the introduction of different temporal scales conducted to a small improvement of the results and, hence, the second method achieved the best Jaccard index in this work, precisely 0.7104. Third, the results obtained with the last method were not the expected ones, since it is known that CNNs have much more capacity than the one here presented. As discussed above, different reasons may appear behind this: undiscriminating data, low number of good examples, overfitting or even a bad choice for the CNN architecture. Despite all, the aim of summarizing and understanding the mechanics behind CNNs was successfully achieved.

There are several directions for future work. As discussed in the start of this dissertation, the efforts of the entire computer vision community are aligned to deep learning, in order to automatically learn features from raw data. Furthermore, with the increase in the availability of dedicated processing units, deep learning models to feature extraction will

become more prevalent. For this reason, in a future work, CNNs can be exhaustively used to perform feature learning, both from skeleton data as from RGB-D images. After that, it would be interesting to verify whether the performance can be improved by the integration of different recognition models. So, also in a future work, the introduction of a Recurrent Neural Network can be tested, possibly with LSTM cells.

BIBLIOGRAPHY

Bo Yang, J., Nhut Nguyen, M., Phyo San, P., Li Li, X. and Krishnaswamy, S. (2015) 'Deep Convolutional Neural Networks On Multichannel Time Series For Human Activity Recognition', in *Proceedings of the 24th International Conference on Artificial Intelligence*. Buenos Aires, Argentina: AAAI Press, pp. 3995–4001.

Camgöz, N. C., Kindiroglu, A. A. and Akarun, L. (2015) 'Gesture Recognition using Template Based Random Forest Classifiers', in Agapito, L., Bronstein, M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 579–594.

Cecotti, H. and Graser, A. (2011) 'Convolutional Neural Networks for P300 Detection with Application to Brain-Computer Interfaces', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3), pp. 433–445.

Chang, J. Y. (2015) 'Nonparametric Gesture Labeling from Multi-modal Data', in Agapito, L., Bronstein, M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 503–517.

Chen, G., Clarke, D., Giuliani, M., Gaschler, A., Weikersdorfer, D. and Knoll, A. (2015) 'Multi-modality Gesture Detection and Recognition With Un-supervision, Randomization and Discrimination', in Agapito, L., Bronstein, M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 608–622.

Chen, X. and Koskela, M. (2013) 'Online RGB-D Gesture Recognition with Extreme Learning Machines', in *Proceedings of the 15th ACM on International conference on multimodal interaction*. Sydney, Australia: ACM Press, pp. 467–474.

Ciresan, D., Meier, U. and Schmidhuber, J. (2012) 'Multi-column Deep Neural Networks for Image Classification', in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition*. Washington DC, USA: IEEE Computer Society, pp. 3642–3649.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K. and Kuksa, P. (2011) 'Natural Language Processing (Almost) from Scratch', *Journal of Machine*

Learning Research, 12, pp. 2493–2537.

Dollár, P., Rabaud, V., Cottrell, G. and Belongie, S. (2005) 'Behavior Recognition via Sparse Spatio-Temporal Features', in *Proceedings of the 14th International Conference on Computer Communications and Networks*. Washington DC, USA: IEEE Computer Society, pp. 65–72.

Escalera, S., Baró, X., Escalante, H. J. and Guyon, I. (2017) 'ChaLearn Looking at People: A Review of Events and Resources'.

Escalera, S., Baró, X., Gonzàlez, J., Bautista, M. A., Madadi, M., Reyes, M., Ponce-López, V., Escalante, H. J., Shotton, J. and Guyon, I. (2015) 'ChaLearn Looking at People Challenge 2014: Dataset and Results', in Agapito, L., Bronstein, M. M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 459–473.

Evangelidis, G., Singh, G. and Horaud, R. (2015) 'Continuous gesture recognition from articulated poses', in Agapito, L., Bronstein, M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 595–607.

Farabet, C., Couprie, C., Najman, L. and LeCun, Y. (2012) 'Scene Parsing with Multiscale Feature Learning, Purity Trees, and Optimal Covers', in Langford, J. and Pineau, J. (eds) *Proceedings of the 29th International Conference on Machine Learning*. New York, USA, pp. 575–582.

Hadsell, R., Sermanet, P., Ben, J., Erkan, A., Scoffier, M., Kavukcuoglu, K., Muller, U. and Lecun, Y. (2009) 'Learning Long-Range Vision for Autonomous Off-Road Driving', *Journal of Field Robotics*, 26(2), pp. 120–144.

Han, J., Shao, L., Xu, D. and Shotton, J. (2013) 'Enhanced Computer Vision with Microsoft Kinect Sensor: A Review', *IEEE Transactions on Cybernetics*. IEEE Computer Society, 43(5), pp. 1318–1334.

Hinton, G. E., Osindero, S. and Teh, Y.-W. (2006) 'A Fast Learning Algorithm for Deep Belief Nets', *Neural computation*, 18(7), pp. 1527–1554.

Hochreiter, S. and Schmidhuber, J. (1997) 'Long Short-Term Memory', *Neural Computation*. MIT Press, 9(8), pp. 1735–1780.

Ji, S., Xu, W., Yang, M. and Yu, K. (2013) '3D Convolutional Neural Networks for Human Action Recognition', *IEEE Transactions on Pattern Analysis and Machine*

Intelligence. IEEE Computer Society, 35(1), pp. 221–231.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R. and Fei-Fei, L. (2014) 'Large-scale Video Classification with Convolutional Neural Networks', in *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, pp. 1725–1732.

Kläser, A., Marszalek, M. and Schmid, C. (2008) 'A Spatio-Temporal Descriptor Based on 3D-Gradients', in Everingham, M., Needham, C., and Fraile, R. (eds) *British Machine Vision Conference*. Leeds.

Krizhevsky, A., Sutskever, I. and Hinton, G. E. (2012) 'ImageNet Classification with Deep Convolutional Neural Networks', in Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds) *Proceedings of the th International Conference on Neural Information Processing Systems*. Lake Tahoe, Nevada: Curran Associates, Inc., pp. 1097–1105.

Laptev, I. and Lindeberg, T. (2006) 'Local Descriptors for Spatio-Temporal Recognition', in MacLean, W. J. (ed.) *Spatial Coherence for Visual Motion Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 91–103.

LeCun, Y., Bengio, Y. and Hinton, G. (2015) 'Deep learning', *Nature*. Nature, 521(7553), pp. 436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. and Jackel, L. D. (1989) 'Backpropagation Applied to Handwritten Zip Code Recognition', *Neural Computation*, pp. 541–551.

Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998) 'Gradient-Based Learning Applied to Document Recognition', *Proceedings of the IEEE*, 86(11), pp. 2278–2324.

Lippmann, R. (1987) 'An Introduction to Computing with Neural Nets', *IEEE ASSP Magazine*, pp. 4–22.

Maas, A. L., Hannun, A. Y. and Ng, A. Y. (2013) 'Rectifier Nonlinearities Improve Neural Network Acoustic Models', in *Proceedings of the 30 th International Conference on Machine Learning*, p. 6.

Maurtua, I. (2015) *Trust and Interaction in Industrial Human-Robot Collaborative applications*.

Mohamed, A., Dahl, G. E. and Hinton, G. (2012) 'Acoustic Modeling using

Deep Belief Networks', *IEEE Transactions on Audio, Speech, and Language Processing*, 20(1), pp. 14–22.

Monnier, C., German, S. and Ost, A. (2015) 'A Multi-scale Boosted Detector for Efficient and Robust Gesture Recognition', in Agapito, L., Bronstein, M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 491–502.

Nandakumar, K., Wan, K. W., Chan, S. M. A., Ng, W. Z. T., Wang, J. G. and Yau, W. Y. (2013) *A Multi-modal Gesture Recognition System Using Audio, Video, and Skeletal Joint Data*.

Neverova, N., Wolf, C., Taylor, G. W. and Nebout, F. (2015a) 'Multi-scale deep learning for gesture detection and localization', in Agapito, L., Bronstein, M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 474–490.

Neverova, N., Wolf, C., Taylor, G. W. and Nebout, F. (2015b) *Multi-scale deep learning for gesture detection and localization*.

Ng, J. Y.-H., Hausknecht, M., Vijayanarasimhan, S., Vinyals, O., Monga, R. and Toderici, G. (2015) 'Beyond Short Snippets: Deep Networks for Video Classification', in *2015 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE Computer Society, pp. 4694–4702.

Ordóñez, F. and Roggen, D. (2016) 'Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition', *Sensors*. Multidisciplinary Digital Publishing Institute, 16(1), p. 115.

Peng, X., Wang, L., Cai, Z. and Qiao, Y. (2015) 'Action and Gesture Temporal Spotting with Super Vector Representation', in Agapito, L., Bronstein, M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 518–527.

Pigou, L., Dieleman, S., Kindermans, P.-J. and Schrauwen, B. (2015) 'Sign Language Recognition using Convolutional Neural Networks', in Agapito, L., Bronstein, M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 572–578.

Schmidhuber, J. (2015) 'Deep learning in neural networks: An overview', *Neural Networks*, 61, pp. 85–117.

Shao, L., Zhen, X., Tao, D. and Li, X. (2014) 'Spatio-Temporal Laplacian Pyramid Coding for Action Recognition', *IEEE Transactions on Cybernetics*, 44(6), pp. 817–827.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. (2014) 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting', *Journal of Machine Learning Research*, 15, pp. 1929–1958.

Taylor, G. W., Fergus, R., LeCun, Y. and Bregler, C. (2010) 'Convolutional Learning of Spatio-temporal Features', in Daniilidis, K., Maragos, P., and Paragios, N. (eds) *Computer Vision - ECCV 2010*. Berlin, Heidelberg: Springer, pp. 140–153.

Wang, H., Kläser, A., Schmid, C. and Liu, C.-L. (2013) 'Dense Trajectories and Motion Boundary Descriptors for Action Recognition', *International Journal of Computer Vision*, 103(1), pp. 60–79.

Wang, H., Ullah, M. M., Klaser, A., Laptev, I. and Schmid, C. (2009) 'Evaluation of local spatio-temporal features for action recognition', in Cavallero, A., Prince, S., and Alexander, D. (eds) *British Machine Vision Conference*. London: British Machine Vision Association.

Weston, J. and Watkins, C. (1999) 'Support Vector Machines for Multi-Class Pattern Recognition', in *Proceedings of the 7th European Symposium on Artificial Neural Networks*, pp. 219–224.

Wu, D. (2015) 'Deep Dynamic Neural Networks for Gesture Segmentation and Recognition', in Agapito, L., Bronstein, M. M., and Rother, C. (eds) *Computer Vision - ECCV 2014 Workshops*. Cham: Springer International Publishing, pp. 552–571.

Wu, D., Pigou, L., Kindermans, P. J., Le, N. D. H., Shao, L., Dambre, J. and Odobez, J. M. (2016) 'Deep Dynamic Neural Networks for Multimodal Gesture Segmentation and Recognition', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(8), pp. 1583–1597.

Wu, D. and Shao, L. (2013) 'Silhouette Analysis-Based Action Recognition Via Exploiting Human Poses', *IEEE Transactions on Circuits and Systems for Video Technology*, 23(2), pp. 236–243.

Yang, M., Ji, S., Xu, W., Wang, J., Lv, F., Yu, K., Gong, Y., Dikmen, M., Lin, D. J. and Huang, T. S. (2009) 'Detecting Human Actions in Surveillance Videos', in *Proc. TREC Video Retrieval Evaluation Workshop*.

Zanfir, M., Leordeanu, M. and Sminchisescu, C. (2013) ‘The Moving Pose: An Efficient 3D Kinematics Descriptor for Low-Latency Action Recognition and Detection’, in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2752–2759.

Zeiler, M. D. and Fergus, R. (2014) ‘Visualizing and Understanding Convolutional Networks’, in Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T. (eds) *Computer Vision - ECCV 2014*. Cham: Springer International Publishing, pp. 818–833.