



DISSERTAÇÃO DE MESTRADO

**HUMAN ACTION RECOGNITION IN IMAGE SEQUENCES  
BASED ON A TWO-STREAM  
CONVOLUTIONAL NEURAL NETWORK CLASSIFIER**

**Vinícius de Oliveira Silva**

**Orientador: Prof. Dr. Alexandre Ricardo Soares Romariz**

**Brasília, Agosto de 2017**

**UNIVERSIDADE DE BRASÍLIA**

FACULDADE DE TECNOLOGIA

**UNIVERSIDADE DE BRASÍLIA  
FACULDADE DE TECNOLOGIA  
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**HUMAN ACTION RECOGNITION IN IMAGE SEQUENCES BASED  
ON A TWO-STREAM CONVOLUTIONAL NEURAL NETWORK  
CLASSIFIER**

**VINICIUS DE OLIVEIRA SILVA**

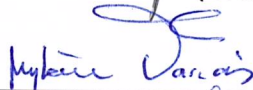
DISSERTAÇÃO DE Mestrado submetida ao Departamento de Engenharia Elétrica da Faculdade de Tecnologia da Universidade de Brasília, como parte dos requisitos necessários para a obtenção do grau de Mestre.

APROVADA POR:



---

ALEXANDRE RICARDO SOARES ROMARIZ, Dr., ENE/UNB  
(ORIENTADOR)



---

MYLENE CHRISTINE QUEIROZ DE FARIAS, Dra., ENE/UNB  
(EXAMINADORA INTERNA)



---

LI WEIGANG, Dr., CIC/UNB  
(EXAMINADOR EXTERNO)

Brasília, 07 de agosto de 2017.

## FICHA CATALOGRÁFICA

SILVA, VINÍCIUS DE OLIVEIRA

HUMAN ACTION RECOGNITION IN IMAGE SEQUENCES BASED ON A TWO-STREAM CONVOLUTIONAL NEURAL NETWORK CLASSIFIER [Distrito Federal] 2017.

xvi, 66 p., 210 x 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2017).

Dissertação de Mestrado - Universidade de Brasília, Faculdade de Tecnologia.

Departamento de Engenharia Elétrica

1. Human action recognition

2. Convolutional neural networks

3. Dense Optical Flow

4. Transfer Learning

I. ENE/FT/UnB

II. Título (série)

## REFERÊNCIA BIBLIOGRÁFICA

SILVA, V.O. (2017). HUMAN ACTION RECOGNITION IN IMAGE SEQUENCES BASED ON A TWO-STREAM CONVOLUTIONAL NEURAL NETWORK CLASSIFIER. Dissertação de Mestrado, Publicação: PPGA-672/2017, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 66 p.

## CESSÃO DE DIREITOS

AUTOR: Vinícius de Oliveira Silva

TÍTULO: HUMAN ACTION RECOGNITION IN IMAGE SEQUENCES BASED ON A TWO-STREAM CONVOLUTIONAL NEURAL NETWORK CLASSIFIER.

GRAU: Mestre em Engenharia de Sistemas Eletrônicos e Automação ANO: 2017

É concedida à Universidade de Brasília permissão para reproduzir cópias desta Dissertação de Mestrado e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. Os autores reservam outros direitos de publicação e nenhuma parte dessa Dissertação de Mestrado pode ser reproduzida sem autorização por escrito dos autores.



Vinícius de Oliveira Silva

Depto. de Engenharia Elétrica (ENE) - FT

Universidade de Brasília (UnB)

Campus Darcy Ribeiro

CEP 70919-970 - Brasília - DF - Brasil

## Acknowledgments

*I first thank God for the wisdom and knowledge that have been given to me.*

*I thank my father, Osmundo Brilhante and his wife Eliana Yukiko Takenaka, for the good advices and for support that was given to me during this work. My grandparents Manoel Alexandre da Silva and Maria José de Oliveira Silva and the other relatives who, even geographically distant, have always been encouraging me.*

*I would also like to thank my teachers and supervisors, Prof. Alexandre Romariz and Prof. Flávio Vidal, for the guidance, the teachings, the understanding and for believing that I was capable of accomplishing this work.*

*To Professor Mylène, also my coordinator during the course of the Master's Degree, for sharing her knowledge and having been so solicitous and patient with me whenever I needed her help. For the computer and room provided by the GPDS / ENE / FT / UnB (Digital Signal Processing Group) that were fundamental for the completion of this work, thank you very much. Also to Professors Daniel (ENE / FT), Antônio (ENE / FT), Li Weigang (CIC / UnB), José Maurício (ENM / FT) whom I had the immense satisfaction of knowing in these two years.*

*I would like to thank the friends that I made during the Masters period that were very important in this journey, at a time when I was far from my family: Wiliam, Pedro Jorge, Wesley, Josua, Henrique, Luiz, Hellard, Mauro and Maíra. To thank and dedicate this work to friends and colleagues who participated more closely in the development of this project: Gustavo, Gizele, Douglas, Guilherme, Frabrizio, Dário and Ana Paula, with whom I was able to exchange many knowledge and I received many advices from.*

*I would also like to thank Prof. Flávio Vidal for the space provided in LISA / CIC / UnB (Laboratory of Images, Signals and Audio) and by the computer "monstro verde" that allowed me to execute my final tests of this project.*

*I would like to thank the staff of FT, CIC and SG11 who, directly or indirectly, contributed to the realization of this project.*

Vinícius de Oliveira Silva

---

## ABSTRACT

The technological evolution in the last decades has contributed to the improvement of computers with excellent processing and storage capacity and cameras with higher digital quality. Nowadays, video generation devices are simpler to manipulate, more portable and with lower prices. This allowed easy generation, storage and transmission of large amounts of videos, which demands a form of automatic analysis, independent of human assistance for evaluation and exhaustive search of videos. There are several applications that can benefit from such techniques such as virtual reality, robotics, tele-medicine, human-machine interface, tele-surveillance and assistance to the elderly in timely caregiving.

This work describes a method for human action recognition in a sequence of images using two convolutional neural networks (CNNs). The Spatial network stream is trained using frames from a sequence of images with transfer learning techniques from the VGG16 network (pre-trained for classification of objects). The other stream channel, Temporal stream, receives stacks of Dense Optical Flow (DOF) as input and it is trained from scratch.

The technique was tested in two public action video datasets: Weizmann and UCF Sports. In the Spatial stream approach we achieve 84.44% of accuracy on Weizmann dataset and 78.46% on UCF Sports dataset. With the Temporal and Spatial streams combined, we obtained an accuracy rate of 91.11% for the Weizmann dataset.

---

## RESUMO

A evolução tecnológica nas últimas décadas contribuiu para a melhoria de computadores com excelente capacidade de processamento, armazenamento e câmeras com maior qualidade digital. Os dispositivos de geração de vídeo têm sido mais fáceis de manipular, mais portáteis e com preços mais baixos. Isso permitiu a geração, armazenamento e transmissão de grandes quantidades de vídeos, o que demanda uma forma de análise automática de informações, independente de assistência humana para avaliação e busca exaustiva de vídeos. Existem várias aplicações que podem se beneficiar de técnicas de inteligência computacional, tais como realidade virtual, robótica, telemedicina, interface homem-máquina, tele-vigilância e assistência aos idosos em acompanhamento constante.

Este trabalho descreve um método para o Reconhecimento de Ações Humanas em sequências de imagens usando duas Redes (canais) Neurais Convolutivas (RNCs). O Canal Espacial é treinado usando quadros de uma sequência de imagens com técnicas de transferência de aprendizagem a partir da rede VGG16 (pré-treinada para classificação de objetos). O outro canal, Canal Temporal, recebe pilhas de Fluxo Óptico Denso (FOD) como entrada e é treinado com pesos iniciais aleatórios.

A técnica foi testada em dois conjuntos de dados públicos de ações humanas: Weizmann e UCF Sports. Na abordagem do Canal Espacial, conseguimos 84,44% de precisão no conjunto de dados Weizmann e 78,46% no conjunto de dados UCF Sports. Com os canais temporal e espacial combinados, obtivemos uma taxa de precisão de 91,11% para o conjunto de dados Weizmann.

# CONTENTS

<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	RELATED WORK	2
1.2	GOALS AND CONTRIBUTIONS	3
1.3	PRESENTATION OF THE MANUSCRIPT	4
<b>2</b>	<b>COMPUTER VISION</b>	<b>5</b>
2.1	OPTICAL FLOW	5
2.1.1	IMAGE AND SEQUENCE OF IMAGES	5
2.1.2	OPTICAL DISPLACEMENT AND OPTICAL FLOW	7
2.1.3	OPTICAL FLOW CALCULATION	10
2.2	FINAL CONSIDERATIONS	19
<b>3</b>	<b>MACHINE LEARNING</b>	<b>20</b>
3.1	CATEGORIES	20
3.2	ARTIFICIAL NEURAL NETWORKS	21
3.2.1	PERCEPTRON	23
3.2.2	MULTILAYER PERCEPTRON	24
3.2.3	ACTIVATION FUNCTIONS	26
3.2.4	ERROR FUNCTIONS	27
3.2.5	OPTIMIZERS	28
3.2.6	THE BACK-PROPAGATION ALGORITHM	28
3.2.7	GENERALIZATION	30
3.3	DEEP LEARNING	31
3.3.1	CNN: CONVOLUTIONAL NEURAL NETWORKS	31
3.3.2	TRANSFER LEARNING	36
3.4	FINAL CONSIDERATIONS	39
<b>4</b>	<b>METHODOLOGY</b>	<b>40</b>
4.1	APPROACH	40
4.2	SPATIAL STREAM	41
4.2.1	SPATIAL STREAM HISTOGRAM	43
4.3	TEMPORAL STREAM	44
4.3.1	STACKS OF DENSE OPTICAL FLOW	44
4.3.2	TEMPORAL STREAM HISTOGRAM	45
4.4	COMBINATION OF CLASSIFIERS	46
<b>5</b>	<b>RESULTS</b>	<b>47</b>
5.1	DATASETS	47

5.2	EXPERIMENTS .....	49
5.2.1	SPATIAL CNN STREAM .....	49
5.2.2	TEMPORAL CNN STREAM .....	50
5.3	EXPERIMENTAL RESULTS .....	51
5.3.1	PRELIMINARY SPATIAL NETWORK .....	51
5.3.2	PRE-TRAINED SPATIAL NETWORK .....	53
5.3.3	TEMPORAL NETWORK .....	53
5.3.4	COMBINATION OF CLASSIFIERS .....	54
<b>6</b>	<b>CONCLUSIONS AND FUTURE WORKS .....</b>	<b>58</b>
	<b>REFERENCES .....</b>	<b>60</b>



## LIST OF FIGURES

2.1	A digital image and the convention used for the pair of axes $x$ and $y$ .....	6
2.2	RGB color image components. Adapted from Gonzalez and Woods [1] .....	6
2.3	Similarity of regions between images, illustrated by Minetto [2].....	7
2.4	(a) and (b) represent a pair of consecutive video frames with the area around a moving hand outlined with a cyan rectangle. (c) represents a close-up of dense optical flow in the outlined area Illustration by Simonyan and Zisserman [3].....	8
2.5	The optical flow equation that defines a line in velocity space, adapted from Beauchemin [4] .....	10
2.6	<i>Aperture</i> problem illustration. Adapted from Minetto [2] .....	10
2.7	Spatial and temporal relations, for the estimation of the partial derivatives of a point of the image, positioned in the center of the cube. Adapted from Horn and Schunck [5]. The index of column $j$ represents the $x$ -axis in the image, the index of row $i$ corresponds to the $y$ -axis and the index $k$ in the time axis. ....	12
2.8	(a) One frame of the Yosemite sequence. (b) The corresponding true velocity field of the image in (a). Illustration by Farnebäck [6].....	17
2.9	(a) and (b) form a pair of consecutive video frames of the two-hands waving movement. (c) is the horizontal component of the displacement vector field, where higher intensity values represent positive values, lower ones represent negative values. (d) Vertical component of the displacement vector field. ....	18
3.1	Anatomy of a neuron. ....	21
3.2	Computational model of a neuron by McCulloch and Pitts [7].....	22
3.3	Perceptron Model.....	23
3.4	Perceptron Multilayer Example .....	25
3.5	Activation functions: Logistic Sigmoid Function (in red), Hyperbolic Tangent Sigmoid Function (in green) with $\gamma = 1$ e $\beta = 1$ and Rectified Linear Function (in blue).....	26
3.6	Properly fitted nonlinear mapping (upper graph) and overfitted nonlinear mapping (lower graph). Adapted from [8].....	30
3.7	(a) represents the second layer (feature maps) (low level features) learned from faces dataset. (b) represents the third layer (feature maps) (higher level features) learned from faces dataset. Both of them illustrated by Lee et al. [9]. ....	32
3.8	The forward propagation of a convolutional layer. ....	33
3.9	The forward propagation of a convolutional layer. ....	34
3.10	An overview of a Convolutional Neural Network. Adapted from LeCun et al. [10].	34
3.11	Dropout illustration. Neurons in white color are dropped out. ....	35
4.1	Proposed Method. ....	42

4.2	Spatial CNN structure: it refers to the Spatial CNN block in the Figure 4.1. The yellow block represents VGG16 convolutional blocks and the green block represents the MLP used. In the yellow block only the fourth and fifth blocks were trained. ....	43
4.3	Histogram of frames classification for Sequence 8 (Weizmann Dataset). The red bar indicates the expected class and the blue bars indicate the other classes.....	43
4.4	Temporal CNN structure: it refers to the Temporal CNN block in the Figure 4.1. ...	44
4.5	Stacks of Dense Optical Flow. ....	45
4.6	Histogram of OF stacks classification for Sequence 8 (Weizmann Dataset). The red bar indicates the expected class and the blue bars indicate the other classes.....	45
5.1	Natural actions performed in the set of image sequences provided by the Weizmann Institute of Science's Computer Vision Laboratory [11].....	48
5.2	Actions performed by video sequences from UCF Sports dataset (illustrated by Soomro and Zamir [12]). ....	49
5.3	Spatial CNN Architecture (Experiment 1).....	49
5.4	(a) Accuracy graphic during training. (b) Loss graphic during training.....	52
5.5	(a) Histogram of frames classification for Sequence 1. (b) Histogram of frames classification for Sequence 4. (c) Histogram of frames classification for Sequence 23. The red bar indicates the expected class and the blue bar incorrect classifications	52
5.6	Confusion matrix of Weizmann dataset using leave-one-out cross-validation. ....	55
5.7	ROC curve of Weizmann dataset. ....	55
5.8	Confusion matrix of UCF Sports dataset using leave-one-out cross-validation. ....	56
5.9	ROC curve of UCF Sports dataset. ....	56

## LIST OF TABLES

3.1	Main differences between L1 and L2 regularization.....	36
3.2	VGG16 Architecture. C denotes a Convolution layer, MP denotes a max pooling layer and FC denotes a fully-connected layer.....	38
5.1	Accuracies on both Weizmann and UCF datasets on each stream.....	53
5.2	Comparison of accuracies on the Weizmann dataset.....	54
5.3	Comparison of accuracies on the UCF dataset .....	57

# LIST OF SYMBOLS

## Symbols

$\Delta$	Variation
$x$	Input data to the neural network
$w$	Weight of a synaptic connection between neurons
$\mathbf{x}$	Input vector or input matrix of the network
$\mathbf{w}$	Weights vector or weights matrix between layers
$b$	Bias of a neuron
$s$	Induced local field of a neuron
$\sigma$	Activation function
$y$	Output of a neuron
$d$	Desired output of the neuron
$D$	Optical flow stack size
$o$	Overlap size between optical flow stacks
$e$	Output error
$\varepsilon$	Error function
$\eta$	Learning rate
$\alpha$	Network <i>momentum</i>
$\delta$	Local gradient of the error function with respect to the weights

## Acronyms

ANN	Artificial Neural Network
CEF	Cross-Entropy Error Function
CNN	Convolutional Neural Network
CNS	Central Nervous System
DL	Deep Learning
MBH	Motion Boundary Histogram
ML	Machine Learning
MLP	Multilayer Perceptron
PNS	Peripheral Nervous System
ReLU	Rectified Linear Unit
RGB	Red Green and Blue
RL	Reinforcement Learning
SEF	Squared Error Function
SGD	Stochastic Gradient Descent
SURF	Speeded-Up Robust Features
SVM	Support Vector Machine

# 1 INTRODUCTION

The technological evolution during the last decades has contributed to the emergence of computers with great processing and storage capacity and high quality digital video cameras. The prices of these video-generating electronics are reducing and they have become portable and simple to manipulate. This allowed society to be able to generate, store and transmit large amounts of media, such as videos. However, analysing the information and content of these videos is not a simple task, often requiring an exhaustive search for the video. There is also a human dependence on the evaluation of these videos, which increases the financial and time costs related to their analysis.

Action recognition in image sequences is a computer vision application in which a set of image processing techniques and time series analysis are used to allow the computer to be able to identify a captured gesture from a camera [13]. The most common application is the automatic translation of sign language into words, but several applications can benefit from these techniques, such as virtual reality, robotics, tele-medicine, man-machine interface and tele-surveillance.

Most of the human action recognition methods in the literature are developed for restricted and short-lived videos that contain simple, well-defined human actions such as waving, running, and jumping [14, 3, 15]. In comparison to the classification of static images, the temporal component of the videos provides an important additional information for the recognition that is based on the information of movement [3].

There are several methods for action recognition based on handcrafted features such as Histograms of Oriented Optical Flow (HOF), Histograms of Oriented Gradient (HOG) and Motion Boundary Histogram (MBH) [16, 17] being used with the Support Vector Machines (SVM) classifier [18, 8].

In order to reduce the effect of camera movement, the Wang and Schmid method [19] produces improved dense trajectories. The estimation of this movement is done using the SURF descriptor [20]. A higher-level representation of the actions was proposed by Sadanand and Corso [21] being combined with a linear SVM classifier.

In this work, we intend to use the Deep Convolutional Neural Networks (CNN), normally used to classify static images, to recognize actions in video data [22]. These networks are used in a Deep Learning (DL) method in which the network is able to find certain patterns through its Convolution and Pooling layers [23]. In the literature, this task has been performed using the frames of the image sequences in a stacked form as input to the network, but did not present good results when compared with results of handcrafted shallow features [14].

In this work, an architecture based on a two-stream CNN was investigated, in which one network receives single frames of the sequence of images (Spatial features). Another one receives stacked dense optical flow components (Temporal features) [24, 25]. Then, those networks were

combined to produce a single output [14, 26]. With this, we expect to generate a robust discriminative model.

## 1.1 RELATED WORK

Research in action recognition has been strongly driven by advances in image recognition methods, which in turn have been adapted to serve video data. In general, action recognition methods are divided into two groups. The first one is the conventional pipeline approach that uses a descriptor followed by a classifier [27, 28, 29, 19, 21, 30] and the second one is a convolutional approach based on deep learned features [22, 31, 32, 33, 34].

The first one models the dynamics of motion in the sequence of images using graphical models or identifying descriptive features [27, 35] through local spatio-temporal features such as Histogram of Oriented Gradients (HOG) [36] and Histogram of Optical Flow (HOF), and then performs classification, often using the SVM [18] classifier (handcrafted features approach). The second one is based on deep convolutional neural networks (CNNs) [10] that can be trained end-to-end (from raw images to labels) in a supervised manner.

There have been some attempts to develop a deep neural network architecture for video recognition. In most of the works, the network received as input a stack of sequenced video frames. With this, the model was expected to implicitly learn spatio-temporal motion features in its first convolutional layers, however this task has proved more difficult than anticipated [3].

Ji *et al.* [34] used CNNs for discriminative end-to-end video learning and a comparison was made between various CNN architectures for action recognition [22]. It was also verified that a network, operating with sequenced frames stacked as input, has similar performance to a network operating in individual video frames. It indicates that spatio-temporal motion features obtained by sequentially stacked frames do not capture enough movement information to discriminate the actions [22].

Ji *at al.* proposed a 3D CNN, so that the features were learned simultaneously in the spatial and temporal dimensions through 3D convolutions [34]. However, in addition to the raw images, a set of hardwired kernels is created to generate the gradients and optical flow that must be learned by the proposed convolutional network, that is, handcrafted features are used. A two-stream CNN was proposed by Simonyan and Zisserman [3], in which each frame of the video is used individually for network training. One stream is fed by raw images and the other by dense optical flow components calculated between consecutive frames. The classification is done by late fusion of the two streams.

The use of pre-trained convolutional models has shown an improvement in classification rates [22, 37, 38]. Karpathy *et al.* [22] analysed architectures in which different convolutional networks were fused in their last completely connected layers for action recognition in a large dataset and for extending the connectivity of a CNN in time domain to take advantage of local

spatio-temporal information.

In this work we use a two-stream convolutional architecture, in which each stream is trained separately. One of them, called Spatial stream network in this work, uses a 2D convolutional architecture named VGG16 [39], pre-trained on the largest and most challenging ILSVRC-2014 dataset, which has achieved good accuracy rates in object recognition. The other one (Temporal stream network) is also a 2D convolutional network which receives stacked dense optical flow components as input.

In the Spatial stream, we use the last convolutional block of the VGG16 network as a single-frame descriptor, and these descriptors feed a multilayer perceptron network. This network is trained with individual frames descriptors from the sequence of images. Next, a fine-tuning is done on the last two convolutional blocks of the VGG16 network, using the previously trained MLP, in order to bootstrap the convergence of the network and reduce possible overfitting during training process. In the case of the Temporal stream network, each sequence of images provides a certain amount of dense optical flow stacks, and during training, we use as training samples all stacks belonging to all image sequences of the training dataset.

After both are trained, we generate histograms that indicate the frequency of frames (for the Spatial stream) and stacks of dense optical flow (for the Temporal stream) assigned to each class. We then normalize these histograms, finally classify the videos (image sequences) by adding spatial and temporal normalized histograms and choosing the most frequent class.

## 1.2 GOALS AND CONTRIBUTIONS

The objective of this project is to develop modules that allow the extraction of features from the optical flow and high level information capable of discriminating human gestures / actions in image sequences; We investigate architectures of Deep Convolutional Neural Networks for action recognition in image sequences, with the challenge of capturing complementary information from the movement between video frames and the appearance of static frames [3].

A two-stream Convolutional Neural Network architecture is proposed, in which one stream is related to the spatial component of the sequence of images and the other to the temporal component. The temporal component is able to achieve good performance with the aid of multi-frame dense optical flow [24]. Then these two structures are combined, so that a better accuracy rate in the classification of the actions is reached.

As contributions, we showed that static frames belonging to a certain sequence of images allow us to classify the action performed in such a sequence. We can also highlight the reuse of pre-trained networks such as the VGG16 network, which was pre-trained for a dataset of 1000 classes of different objects and some actions are associated with certain types of objects. This transfer learning contributed significantly to the learning of the spatial network.



### 1.3 PRESENTATION OF THE MANUSCRIPT

Chapter 2 describes the main techniques of optical flow calculation, as well as the Farneback Algorithm, which was used in this work. Chapter 3 briefly discusses the area of Machine Learning, explains the most important concepts and algorithms on Artificial Neural Networks (ANNs), emphasizing the Convolutional Neural Networks (CNNs) that formed the basis of the classifier developed in this work. Chapter 3 also presents neural network regularization techniques and a brief explanation of Transfer Learning.

Chapter 4 presents the methodology used for our human actions classifier, based on two streams, a spatial network (Spatial CNN stream) and a temporal one (Temporal CNN stream) and how the classification is made from their combination. In Chapter 5 we present the datasets used for the experiments of this work, explain the experimental setup and present the respective results. We conclude the paper in Chapter 6 by presenting our contributions and suggesting ideas for future work.

## 2 COMPUTER VISION

Computer vision is a field that develops methods for acquisition, processing and analysis of images and sequence of images (videos). Its purpose is to produce numeric or symbolic results to be used by electronic systems [40]. This field also attempts to give human vision dual ability by means of the understanding and perception of images electronically [41]. Such an understanding of image can be seen as a transformation of symbolic information from image data into representations for human perception. All this has been done using models developed with the aid of geometry, statistics, and learning theory [42].

Computer vision systems try to automate tasks that the human visual system is capable of performing. Its sub-fields include scene understanding, event detection, video tracking, image restoration, object recognition and motion estimation.

This chapter provides a brief background on computer vision and optical flow algorithms. In the following sections, the main concepts of computer vision used for the development of this dissertation will be introduced to the reader.

### 2.1 OPTICAL FLOW

The calculation of Optical Flow is a Computer Vision technique normally used to estimate the motion between two frames of a sequence of images. This method is able to estimate the movement that occurs in the sequence of images without prior knowledge of the content of these images. This section presents respectively the concepts of images, sequence of images and Optical Flow, as well as the main algorithms used to calculate the latter.

#### 2.1.1 Image and Sequence of Images

An image is defined as a two-dimensional luminous intensity function, mathematically described by a function  $f(x, y)$  at any point of spatial coordinates  $(x, y)$  and its value is proportional to the brightness (or gray level) of the image in that point [1]. The image intensity levels are often called gray levels when those images are monochromatic (monochromatic light), that is, they have only one channel.

In the case of images that have separate information in different frequency bands, a function  $f(x, y)$  is required for each band. This is the case for RGB standard colored images, which are formed by the information of additive primary colors, such as red (R - Red), green (G - Green) and blue (B - Blue) [1].

In this work, we will use digital images. To digitize an image, its coordinates and intensity



Figure 2.1: A digital image and the convention used for the pair of axes  $x$  and  $y$ .

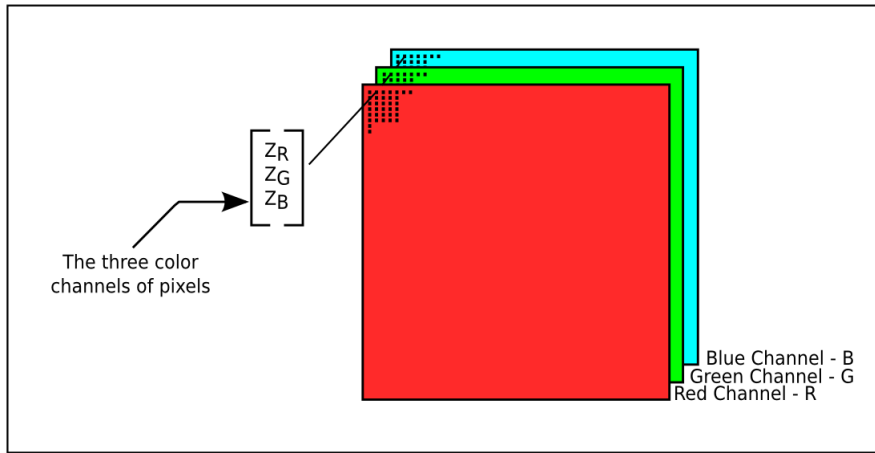


Figure 2.2: RGB color image components. Adapted from Gonzalez and Woods [1]

levels must be digitized. Then, the coordinate values are sampled and amplitude values of the image are quantized. With the values of  $x$ ,  $y$  and the amplitude of  $f(x, y)$  in discrete and finite quantities, we can say that the image is digital and coordinates  $(x, y)$  represent a pixel. In the course of this dissertation, we will represent a digital image as shown in the following Equation:

$$f(x, y) = \begin{bmatrix} f(0, 0) & f(0, 1) & \cdots & f(0, N - 1) \\ f(1, 0) & f(1, 1) & \cdots & f(1, N - 1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M - 1, 0) & f(M - 1, 1) & \cdots & f(M - 1, N - 1) \end{bmatrix}, \quad (2.1)$$

where  $M$  and  $N$  denote, in this order, the number of rows and columns of the digital image resulting from the discretization of  $f(x, y)$ ,  $x = 0, 1, 2, \dots, M - 1$ ,  $y = 0, 1, 2, \dots, N - 1$  and  $M \times N$  represents the size of this image.

A sequence of images is a three-dimensional function  $h(x, y, t)$  which has two or more images, such as  $f_1(x, y)$ ,  $f_2(x, y)$ ,  $\dots$ ,  $f_n(x, y)$ , taken at discrete time instants represented by  $t$ .

## 2.1.2 Optical Displacement and Optical Flow

We present below a definition of optical correspondence between images by Minetto [2]. In this work we will call it similarity of regions between images. Given a pixel  $p$  belonging to a digital image  $J$ , we can define its corresponding pixel  $q$  in another image  $K$ , so that the similarity between the pixel values of the image  $J$  in the neighborhood of  $p$  and the image  $K$  in the neighborhood of  $q$  are maximized. One can observe in Figure 2.3 the illustration of this concept.

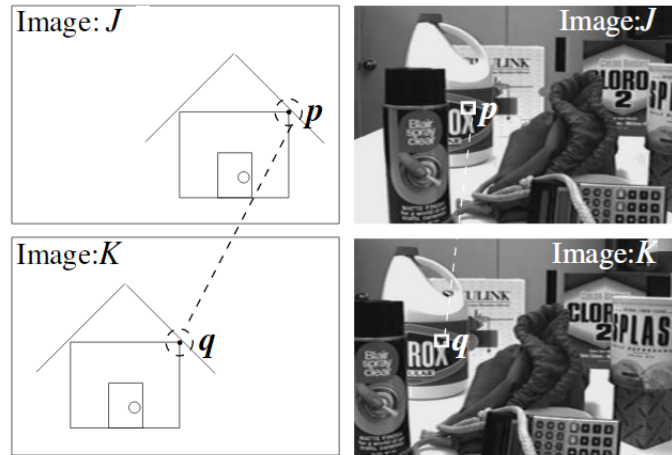


Figure 2.3: Similarity of regions between images, illustrated by Minetto [2]

In general, this concept of similarity applies to two images belonging to a sequence of images, both of which present the same scene in different moments of time, due to the variation of the position of the objects in the scene or to the camera movement. Similarly to Minetto [2], we will denote mathematically the similarity of regions between images by

$$J(p) \approx K(q). \quad (2.2)$$

If  $p$  and  $q$  are similar pixels, we say that the displacement of  $p$  in the image  $J$  to the position  $q$  in the image  $K$ , in the spatial domain of the image, is given by the vector  $\mathbf{f} = q - p$ . In this way, we define that the Optical Flow of an image  $J$  for a posterior image  $K$  in a sequence of images is a function that maps each point  $p$  to its displacement vector  $\mathbf{f}(p)$  and has the following property:

$$J(p) \approx K(p + \mathbf{f}(p)). \quad (2.3)$$

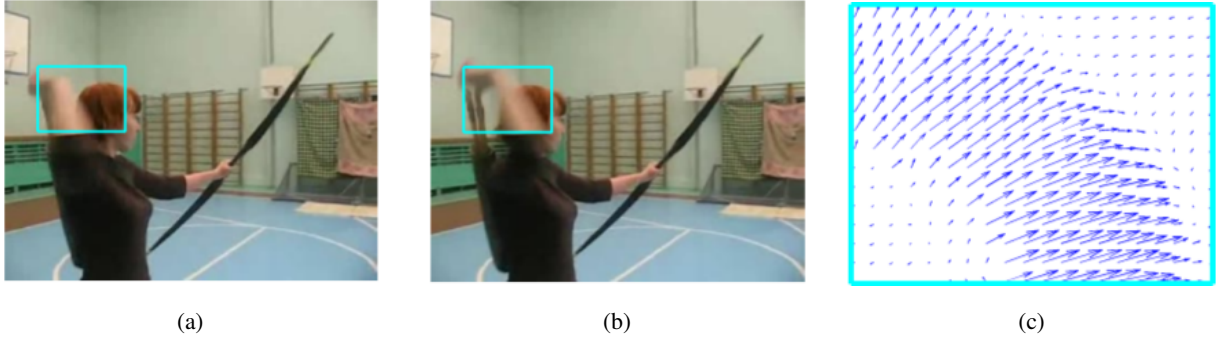


Figure 2.4: (a) and (b) represent a pair of consecutive video frames with the area around a moving hand outlined with a cyan rectangle. (c) represents a close-up of dense optical flow in the outlined area Illustration by Simonyan and Zisserman [3].

An illustration of an Optical Flow  $\mathbf{f}$  is presented in Figure 2.4. It was sampled in a set of points,  $p_1, p_2, \dots, p_n$ , and produced the displacement vectors  $\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_n$ , in which each vector  $\mathbf{f}_i$  has origin in  $p_i$ , same direction of  $\mathbf{f}_i$  and length proportional to its magnitude.

To calculate the Optical Flow, we assume that the intensity of a particular region of an image remains approximately constant in a short period of time, as it changes over time. With this, we assume that the intensity of the corresponding pixels in a sequence of images remains constant. Beauchemin and Barron [4] present a mathematical definition for this hypothesis, considering a continuous sequence of images  $H$ , containing the images  $J$  and  $K$ , as indicated by:

$$J(p) = H(p, t) \quad (2.4)$$

and

$$K(q) = H(p + \mathbf{f}(p), t + \delta t), \quad (2.5)$$

where  $\delta t$  corresponds to a very short time interval. Then, since  $x$  and  $y$  are the coordinates of  $p$  and  $f_x$  and  $f_y$  are the vertical and horizontal components of the displacement  $\mathbf{f}(p)$  between the time instants  $t$  and  $t + \delta t$ , we can rewrite Equation 2.2 as

$$H(x, y, t) = H(x + \delta x, y + \delta y, t + \delta t), \quad (2.6)$$

once

$$J(p) = H(x, y, t) \quad (2.7)$$

and

$$K(q) = H(x + \delta x, y + \delta y, t + \delta t) \quad (2.8)$$

where  $\delta x$  and  $\delta y$  are respectively the individual displacements of the  $x$  and  $y$  coordinates in the image region  $(x, y, t)$ , after a duration of time  $\delta t$ . By expanding the right-hand side of Equation 2.6 by the Taylor series in relation to  $\delta x$ ,  $\delta y$  and  $\delta t$ , we obtain:

$$H(x, y, t) = H(x, y, t) + \delta x \frac{\partial H(x, y, t)}{\partial x} + \delta y \frac{\partial H(x, y, t)}{\partial y} + \delta t \frac{\partial H(x, y, t)}{\partial t} + O(\delta t^2), \quad (2.9)$$

where  $\frac{\partial H(x, y, t)}{\partial x}$ ,  $\frac{\partial H(x, y, t)}{\partial y}$  and  $\frac{\partial H(x, y, t)}{\partial t}$  are respectively the partial derivatives of  $H$  with respect to  $x$ ,  $y$  and  $t$ , and  $O(\delta t^2)$  is the second-order term, generally dismissed as negligible. By subtracting  $H(x, y, t)$  on both sides, neglecting  $O(\delta t^2)$ , omitting  $(x, y, t)$  and dividing Equation 2.9 on both sides by  $\delta t$ , we can rewrite it as:

$$\frac{\partial H}{\partial x} \frac{\delta x}{\delta t} + \frac{\partial H}{\partial y} \frac{\delta y}{\delta t} + \frac{\partial H}{\partial t} = 0. \quad (2.10)$$

Considering  $u = \frac{\delta x}{\delta t}$  and  $v = \frac{\delta y}{\delta t}$  and using  $H_x$ ,  $H_y$  and  $H_t$  for the partial derivatives of the image brightness with respect to  $x$ ,  $y$  and  $t$ , in that order, we obtain a simple linear equation with only two unknown variables as indicated below:

$$H_x u + H_y v + H_t = 0. \quad (2.11)$$

The vector  $\mathbf{v} = (u, v)$  represents the differential Optical Flow in a certain pixel of the image or, more specifically, the instantaneous velocity of a pixel  $(x, y)$  at time  $t$ . Equation 2.11, also known as the Optical Flow constraint equation, can be rewritten as

$$(H_x, H_y) \cdot (u, v) = -H_t, \quad (2.12)$$

where  $(H_x, H_y)$  represents the spatial gradient of  $H$  at time  $t$  and defines a restriction at velocity  $\mathbf{v}$ , as shown in Figure 2.5.

In Figure 2.5, the normal velocity vector  $\mathbf{v}_\perp$  is defined as the vector perpendicular to the constraint line, which is the velocity with the smallest modulus in the Optical Flow constraint line. Due to this constraint, it is not possible to calculate the two components of  $\mathbf{v}$ , since the vector has two components and only one constraint equation was obtained, and it is therefore possible to estimate only the component in the direction of the local gradient of the intensity function of image. We call this problem Optical Flow *aperture* problem. As stated by Beauchemin and Barron in [4], the motion can be estimated from the Optical Flow constraint equation only in regions of the image where there are well-structured information of intensity.

In Figure 2.6, adapted from Minetto [2], we illustrate the *aperture* problem, where each point  $p$  in the first image may have several options of point  $q$  in the second image that have equal or close local similarities. The scene has a square in uniform color moving diagonally upwards, in which its initial position is illustrated with full line and the final position with dashed line. To indicate the local similarity of the most similar region and the Optical Flow of the points, circular windows were used, indicated with full lines in the source image and with dashed lines in the destination image.

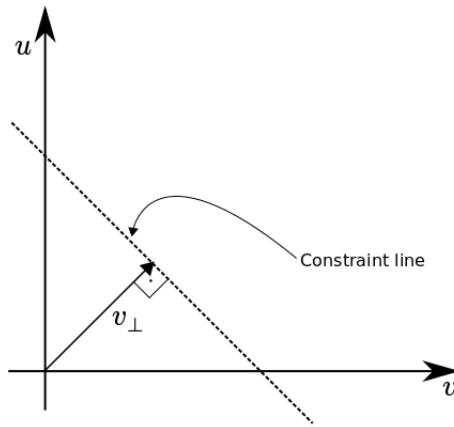


Figure 2.5: The optical flow equation that defines a line in velocity space, adapted from Beauchemin [4]

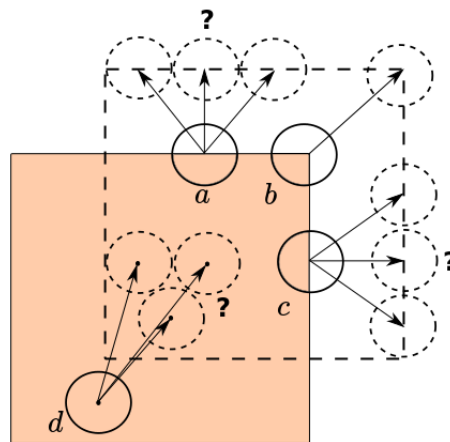


Figure 2.6: *Aperture* problem illustration. Adapted from Minetto [2]

We can see in Figure 2.6 that, for points  $a$  and  $c$ , the ambiguity of similarity of regions between images extends along a line, whereas for point  $b$  the displacement is well defined. For point  $d$ , the ambiguity lies on a region.

In this way, we can estimate only the motion component in the direction of the local gradient of the intensity function of an image. Many algorithms were proposed in order to overcome the *aperture* problem. Some of these will be presented in the next subsection.

### 2.1.3 Optical Flow Calculation

The Optical Flow calculation consists of calculating a displacement representing the distance of a pixel that moved between the previous frame and the current frame of a sequence of images or, equivalently, associating a velocity vector with each pixel of an image.

McCane *et al.* [43], Barron *et al.* [44], and Galvin *et al.* [45] have conducted comparative studies on the performance of various techniques of Optical Flow Calculation, which can be

categorized in differential, energy-based, phase-based and region-based methods.

In the following subsection, we describe the concepts related to the estimation of the partial derivatives of an image point, which are fundamental for the understanding of the calculation algorithms of the Optical Flow. Then, we present the algorithm by Lucas and Kanade [46, 47], which calculates the differential Optical Flow and the algorithm proposed by Horn and Schunck [5], which seeks to meet the criteria of gradient and global smoothness.

### 2.1.3.1 Estimative of Partial Derivatives

For the calculation of the Optical Flow, we must have a consistent estimate of the partial derivatives. The estimation of the partial derivatives is made from a discrete set of measures of intensities. Horn and Schunck [5] present an equation that relates the changes of intensity of a pixel in an image to the movement of its intensity pattern.

As we defined above,  $H(x, y, t)$  represents the intensity of an image in the pixel  $(x, y)$  taken at the discrete time instant  $t$ . When a pattern moves, the intensity of a particular point will remain constant, so that:

$$\frac{dH}{dt} = 0. \quad (2.13)$$

Applying the chain rule, we obtain the following equation:

$$\frac{\partial H}{\partial x} \frac{dx}{dt} + \frac{\partial H}{\partial y} \frac{dy}{dt} + \frac{\partial H}{\partial t} = 0. \quad (2.14)$$

Considering

$$u = \frac{dx}{dt}, \quad (2.15)$$

$$v = \frac{dy}{dt}, \quad (2.16)$$

$$H_x = \frac{\partial H}{\partial x}, \quad (2.17)$$

$$H_y = \frac{\partial H}{\partial y}, \quad (2.18)$$

and

$$H_t = \frac{\partial H}{\partial t}, \quad (2.19)$$

we can rewrite Equation 2.14 as:

$$H_x u + H_y v + H_t = 0. \quad (2.20)$$

Thus, we have a simple linear equation with only two unknown variables  $u$  and  $v$ , which are components of the pixel motion vector (or its Optical Flow). In order to estimate the partial



derivatives  $H_x$ ,  $H_y$  and  $H_t$  of a given point, Horn and Schunck [5] use a cube in which such point is located in its center, formed by eight measures of intensity, whose spatial and temporal relation are represented in Figure 2.7. Each estimate of the partial derivatives is a mean of the differences of the four adjacent measures of the cube, as can be seen in the equations below:

$$H_x \approx \frac{(H_{i,j+1,k} - H_{i,j,k}) + (H_{i+1,j+1,k} - H_{i+1,j,k}) + (H_{i,j+1,k+1} - H_{i,j,k+1}) + (H_{i+1,j+1,k+1} - H_{i+1,j,k+1})}{4}, \quad (2.21)$$

$$H_y \approx \frac{(H_{i+1,j,k} - H_{i,j,k}) + (H_{i+1,j+1,k} - H_{i,j+1,k}) + (H_{i+1,j,k+1} - H_{i,j,k+1}) + (H_{i+1,j+1,k+1} - H_{i,j+1,k+1})}{4}, \quad (2.22)$$

$$H_t \approx \frac{(H_{i,j,k+1} - H_{i,j,k}) + (H_{i+1,j,k+1} - H_{i+1,j,k}) + (H_{i,j+1,k+1} - H_{i,j+1,k}) + (H_{i+1,j+1,k+1} - H_{i+1,j+1,k})}{4}, \quad (2.23)$$

in which the index of column  $j$  represents the  $x$ -axis in the image, the index of row  $i$  corresponds to the  $y$ -axis and the index  $k$  in the time axis.

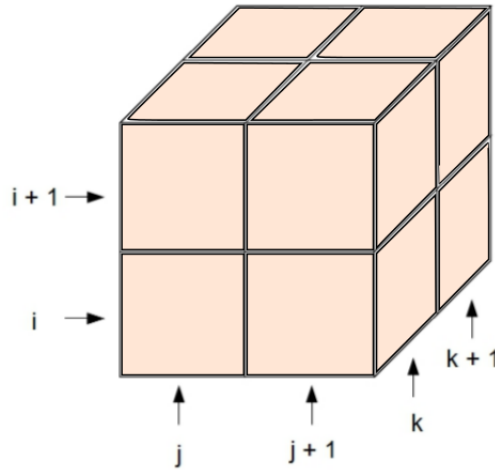


Figure 2.7: Spatial and temporal relations, for the estimation of the partial derivatives of a point of the image, positioned in the center of the cube. Adapted from Horn and Schunck [5]. The index of column  $j$  represents the  $x$ -axis in the image, the index of row  $i$  corresponds to the  $y$ -axis and the index  $k$  in the time axis.

### 2.1.3.2 Horn and Schunck Algorithm

Horn and Schunck proposed a method in 1981 that had a restriction, in order to obtain the two components of the vector  $\mathbf{v}(u, v)$ , capable of producing a consistent system with unique solution, two equations and two unknown variables. In order to overcome the *aperture* problem, Horn and Schunck [5] hypothesized that Optical Flow  $\mathbf{v}(x, y)$  varies smoothly with the position  $(x, y)$ , that is, neighboring pixels have similar movements, generating an uniform flow. Thus, the goal of the

Horn and Schunck algorithm is to minimize the total energy given by:

$$\varepsilon_{total}^2 = \int \int (\alpha^2 \varepsilon_{reg}^2 + \varepsilon_{flow}^2) dx dy, \quad (2.24)$$

where the term  $\varepsilon_{reg}$  is calculated by:

$$\varepsilon_{reg} = \left( \frac{\partial u}{\partial x} \right)^2 + \left( \frac{\partial u}{\partial y} \right)^2 + \left( \frac{\partial v}{\partial x} \right)^2 + \left( \frac{\partial v}{\partial y} \right)^2, \quad (2.25)$$

and corresponds to the spatial variation of the Optical Flow which, in turn, is weighted by  $\alpha^2$ , depending on the quantization error of the image and the noise level present in the sequence of images. It determines the influence of the regularity constraint on minimization. The term  $\varepsilon_{flow}$  is calculated according to:

$$\varepsilon_{flow} = H_x u + H_y v + H_t, \quad (2.26)$$

being related to the error of the changes in the intensity rate in the image, which do not respect the Optical Flow constraint indicated by Equation 2.11.

The problem of minimization of total energy  $\varepsilon_{total}^2$  is reduced to solving a set of differential equations, as stated by the following equations:

$$H_x^2 u + H_x H_y v = \alpha^2 \nabla^2 u - H_x H_t, \quad (2.27)$$

and

$$H_x H_y u + H_y^2 v = \alpha^2 \nabla^2 v - H_y H_t. \quad (2.28)$$

Applying the Laplacian approximation on these equations, we obtain the following system:

$$(\alpha^2 + H_x^2) u + H_x H_y v = (\alpha^2 \bar{u} - H_x H_t) \quad (2.29)$$

and

$$H_x H_y u + (\alpha^2 + H_y^2) v = (\alpha^2 \bar{v} - H_y H_t). \quad (2.30)$$

This system can be rewritten as:

$$u = \bar{u} - \frac{H_x (H_x \bar{u} + H_y \bar{v} + H_t)}{\alpha^2 + H_x^2 + H_y^2} \quad (2.31)$$

and

$$v = \bar{v} - \frac{H_y (H_x \bar{u} + H_y \bar{v} + H_t)}{\alpha^2 + H_x^2 + H_y^2}. \quad (2.32)$$

In order to minimize the Equations 2.31 and 2.32, Horn and Schunck [5] suggest an iterative solution, according to Equations 2.33 and 2.34, where  $\bar{u}^n$  and  $\bar{v}^n$  are the mean velocities of the neighbors of  $u$  and  $v$  in iteration  $n$ .

$$u^{n+1} = \bar{u}^n - \frac{H_x(H_x\bar{u}^n + H_y\bar{v}^n + H_t)}{\alpha^2 + H_x^2 + H_y^2} \quad (2.33)$$

$$v^{n+1} = \bar{v}^n - \frac{H_y(H_x\bar{u}^n + H_y\bar{v}^n + H_t)}{\alpha^2 + H_x^2 + H_y^2} \quad (2.34)$$

### 2.1.3.3 Lucas and Kanade Algorithm

Lucas and Kanade [46, 47] presented another proposal to solve the *aperture* problem of the Optical Flow. The method of calculation of the Optical Flow proposed by Lucas and Kanade [46, 47] assumes three hypotheses:

- Constancy of intensity, that is, any pixel of any object in a scene of a sequence of images will not change its appearance as it moves from frame to frame;
- Temporal persistence (or small movements). That is, objects in the images do not move too much from one frame to another;
- Spatial coherence (or rigidity in movement). Neighboring pixels in a scene, which belong to the same surface, have similar movements.

Based on the first hypothesis, we also use Equation 2.11. Isolating the partial derivative of  $t$ , we can rewrite it so that

$$H_x u + H_y v = -H_t. \quad (2.35)$$

Considering the constraint of spatial coherence, Lucas and Kanade defined similarity in a two-dimensional neighborhood, called the integration window. As shown by Bouguet [48], considering that  $I$  and  $J$  are two images of a sequence of images  $H$ , where  $p = [p_x \ p_y]^T$  is a point on the image  $I$ ,  $q = [q_x \ q_y]^T$  is a point on the image  $J$  and that  $\mathbf{v} = [u \ v]^T$  is the velocity vector (or Optical Flow) of the point  $p$ , the objective of the Optical Flow calculation algorithm is to find the location of  $q = p + v = [p_x + u \ p_y + v]^T$  in the image  $J$ , so that  $I(p) \approx J(q)$ . Thus, vector  $\mathbf{v}$  is defined as the vector that minimizes the residual function, as shown in Equation 2.36:

$$\epsilon(\mathbf{v}) = \epsilon(u, v) = \sum_{x=p_x-l_x}^{p_x+l_x} \sum_{y=p_y-l_y}^{p_y+l_y} (I(x, y) - J(x + u, y + v))^2, \quad (2.36)$$

in which  $l_x$  and  $l_y$  are integer values that define the size of the integration window  $W = (2l_x + 1) \times (2l_y + 1)$ , where the measure of similarity  $I(p) \approx J(q)$  is calculated.

The calculation of the Optical Flow of a pixel  $p$  is done using the system of equations formed by the pixels around it, belonging to the integration window  $W$ . Given that  $r = 2l_x + 1$  and  $s = 2l_y + 1$ , we can calculate the motion of  $p$  from  $n = rs$  equations:

$$\begin{aligned}
H_{x1}u + H_{y1}v &= -H_{t1} \\
H_{x2}u + H_{y2}v &= -H_{t2} \\
H_{x3}u + H_{y3}v &= -H_{t3} \\
&\vdots \quad \quad \quad \vdots \\
H_{xn}u + H_{yn}v &= -H_{tn},
\end{aligned} \tag{2.37}$$

which can be rewritten in its matrix form as:

$$\begin{bmatrix} H_{x1} & H_{y1} \\ H_{x2} & H_{y2} \\ H_{x3} & H_{y3} \\ \vdots & \vdots \\ H_{xn} & H_{yn} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = - \begin{bmatrix} H_{t1} \\ H_{t2} \\ H_{t3} \\ \vdots \\ H_{tn} \end{bmatrix} \tag{2.38}$$

or

$$A\mathbf{v} = -b. \tag{2.39}$$

Thus, we obtain a system with two variables, with more than two equations, making the system consistent with more than one solution. We can use the least squares method to find a solution with minimal error. In its standard form  $\|Ad - b\|^2$  is solved as follows:

$$(A^T A)d = A^T(-b), \tag{2.40}$$

where we calculate the Optical Flow components  $u$  and  $v$ :

$$\mathbf{v} = \begin{bmatrix} u \\ v \end{bmatrix} = (A^T A)^{-1} A^T(-b), \tag{2.41}$$

where

$$A^T A = \begin{bmatrix} \sum H_x H_x & \sum H_x H_y \\ \sum H_x H_y & \sum H_y H_y \end{bmatrix} \tag{2.42}$$

and

$$A^T(-b) = - \begin{bmatrix} \sum H_x H_t \\ \sum H_y H_t \end{bmatrix}. \tag{2.43}$$

The problem can be solved when  $A^T A$  has an inverse matrix, that is, when it has full rank (matrix with two large eigenvectors), which means that the sequence of images  $H$  has significant

gradient in the neighborhood  $W$  of the pixel that is being analysed [2, 49]. The existence of details in the neighborhood  $W$  allows the inversion of the  $A^T A$  matrix and the calculation of the Optical Flow. The absence of details, with regions of constant brightness or where the brightness varies only in one direction, render the matrix null and impossible to calculate the Optical Flow of the points in that region.

The algorithms by Horn and Schunck [5] and Lucas and Kanade [47] consider only small movements. If objects are moving fast, the pixels will move very fast, and the spatial masks of the spatial derivatives will fail. In order to achieve good accuracy we tend to choose small integration windows to maintain the details of the image. The robustness of the algorithms is related to the sensitivity of tracking to changes in the levels of image intensity and to the extent of motion in the image. Therefore, it is common to prefer larger integration windows for larger movements.

In order to establish a model that presented high robustness, a pyramidal implementation of the classic Lucas and Kanade algorithm, the KLT matching algorithm, was proposed by Lucas and Kanade [47] and developed by Tomasi and Kanade [50]. In summary, the KLT algorithm identifies good features to track and then returns indications of how well tracking of each point is occurring.

#### 2.1.3.4 The Dense Optical Flow Algorithm

The two-frame motion estimation based on polynomial expansion is a method proposed in 2003 by Farneback [6] to calculate dense optical flow in real time.

Figure 2.8 presents the displacement vector of each pixel, but such displacement does not appear to occur in certain regions of the image while the flow is determined for the neighborhood of the pixel. Then the objective of the algorithm is to approximate some neighborhood of each pixel with a polynomial, as stated by:

$$f(\mathbf{p}) \sim \mathbf{p}^T \mathbf{A} \mathbf{p} + \mathbf{b}^T \mathbf{p} + c, \quad (2.44)$$

where  $\mathbf{A}$  is a symmetric matrix,  $\mathbf{b}$  a vector,  $c$  is a scalar and  $\mathbf{p}$  represents the pixels of the frame. The coefficients of the polynomial are estimated from a least squares fit to the signal values in the neighborhood, based on the normalized convolution [51].

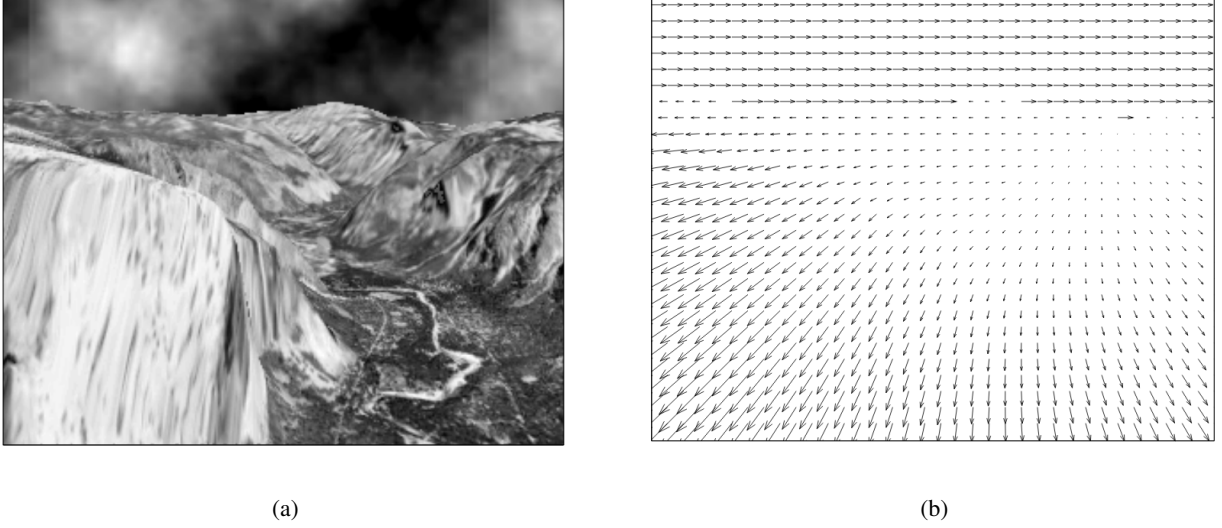


Figure 2.8: (a) One frame of the Yosemite sequence. (b) The corresponding true velocity field of the image in (a). Illustration by Farneback [6]

The displacement between frames is determined by finding such an estimate for the two consecutive frames. After polynomial expansion, each neighborhood is approximated by a second-order polynomial, according to the Equations 2.45 and 2.46, respectively for frames 1 and 2.

$$f_1(\mathbf{p}) = \mathbf{p}^T \mathbf{A}_1 \mathbf{p} + \mathbf{b}_1^T \mathbf{p} + c_1 \quad (2.45)$$

$$f_2(\mathbf{p}) = f_1(\mathbf{p} - \mathbf{d}) = (\mathbf{p} - \mathbf{d})^T \mathbf{A}_1 (\mathbf{p} - \mathbf{d}) + \mathbf{b}_1^T (\mathbf{p} - \mathbf{d}) + c_1 = \mathbf{p}^T \mathbf{A}_2 \mathbf{p} + \mathbf{b}_2^T \mathbf{p} + c_2 \quad (2.46)$$

The matrix  $\mathbf{A}$  gives us information about the even part of the signal,  $\mathbf{b}$  over the odd part, and  $c$  represents the local DC level. By calculating the neighborhood polynomials in the two subsequent images, we can obtain the displacement  $\mathbf{d}$ , in the case of the ideal translation. Observing Equations 2.45, 2.46, 2.47, 2.48 and 2.49, we can see how the coefficients of the polynomial of the second frame are connected to those of the first one and also to the displacement  $\mathbf{d}$  between them.

$$\mathbf{A}_2 = \mathbf{A}_1, \quad (2.47)$$

$$\mathbf{b}_2 = \mathbf{b}_1 - 2\mathbf{A}_1 \mathbf{d} \quad (2.48)$$

and

$$c_2 = \mathbf{d}^T \mathbf{A}_1 \mathbf{d} - \mathbf{b}_1^T \mathbf{d} + c_1 \quad (2.49)$$

From Equation 2.48, it is possible to find  $\mathbf{d}$ , as we can see in Equation 2.50.

$$\mathbf{d} = -\frac{1}{2} \mathbf{A}_1^{-1} (\mathbf{b}_2 - \mathbf{b}_1) \quad (2.50)$$

In practice,  $\mathbf{d}$  contains an error  $e$  which can be minimized by refinement. The steps of the  $e$  refinement process are:

- We take the average between matrixes  $\mathbf{A}_1$  and  $\mathbf{A}_2$  because they are different in practice;
- We assume that  $\mathbf{d}$  is varying slowly and so the information by a neighborhood of each pixel can be integrated;
- Parametrizing the displacement field to a motion model [51];
- Incorporating a priori knowledge through a pyramidal structure similar to that one described by KLT algorithm [50];

The results of the optical flow calculations on a scale provide an approximate estimate of the optical flow for the next processed scale. The polynomials need to be computed for each scale, since the pyramidal structure (multi-scale) is being used.

The Figure 2.9 presents the horizontal and vertical components of the displacement vector field.

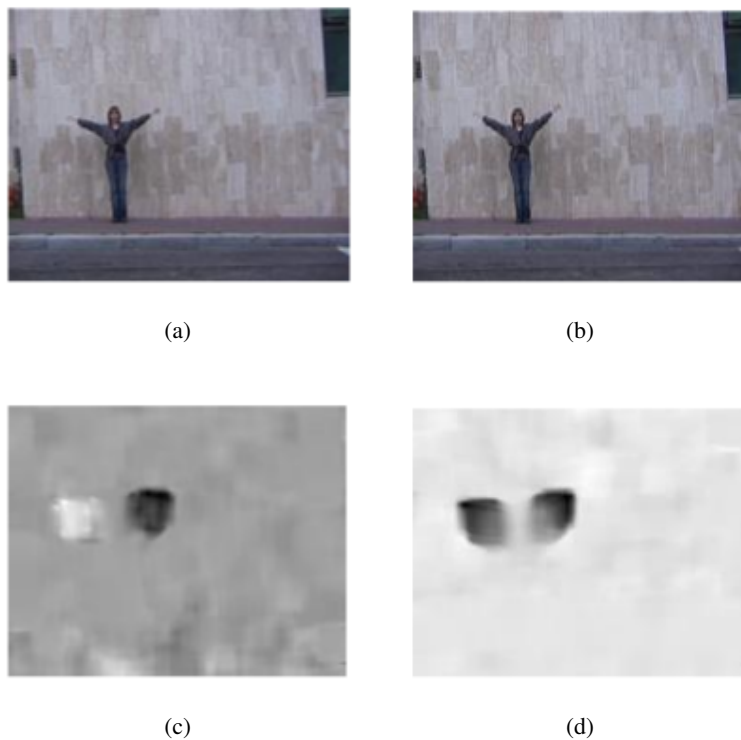


Figure 2.9: (a) and (b) form a pair of consecutive video frames of the two-hands waving movement. (c) is the horizontal component of the displacement vector field, where higher intensity values represent positive values, lower ones represent negative values. (d) Vertical component of the displacement vector field.

## 2.2 FINAL CONSIDERATIONS

The Denso Optical Flow (DOF) calculation is a Computer Vision stage. Its importance consists in the temporary information of movement that occurs in the sequence of images, being this information relevant to the analysis and the recognition of movements (actions). In this Chapter we described the best known techniques of Optical Flow and the technique of Farnebäck [6], used in this work. In the next Section, we will talk about Machine Learning (ML) and Convolutional Neural Networks (CNNs) that were used to develop this work.



## 3 MACHINE LEARNING

Machine learning is a field of data analysis that automates the development of analytical models. Through algorithms that learn iteratively from data, machine learning allows computers to find relevant features in these data without explicit programming [52].

This field was born from pattern recognition and the idea that computers should be capable of learning without being programmed for specific tasks. The iterative character of machine learning is important because models are exposed to new data, and such models are able to adapt independently. They learn from previous calculations to obtain reasonably reliable results [8]. This mechanism works exactly as Alan Turing proposed in his article *Computing Machinery and Intelligence* in which he also proposed to change the question "Can machines think?" for "Can machines do what we (as thinking entities) Can?" [53].

While many machine learning algorithms have been proposed long time ago, their ability to perform complex iterative calculations on large data, with enough speed, is a recent development. This is because computational processing has become cheaper and more powerful and there is affordable data storage.

Machine learning is a multidisciplinary field. It relies on results from artificial intelligence, control theory, information theory, probability and statistics, computational complexity theory, philosophy, psychology, neurobiology and other fields [52].

### 3.1 CATEGORIES

We can categorize the learning processes through which machine learning algorithms work as follows: learning from a teacher (supervised learning) and learning without a teacher. This latter form of learning can be subdivided into unsupervised learning and reinforcement learning [8, 54].

Such categories are:

**Supervised Learning:** It is a machine learning approach in which a function is inferred from supervised training data. That is, a training data set in which each sample is formed by an input object and the desired output value (called a label) for the system. The algorithm analyses the training data and produces a function that is able to generalize a classification rule from training data.

**Unsupervised Learning:** Unsupervised learning consists of analysing data and looking for patterns. It is an extremely powerful tool for identifying the nature of the data. The system has to discover relationships, patterns, regularities or categories, by itself, in the data presented to it and to encode them in the outputs.

**Reinforcement Learning:** It is learning based on interaction with the environment. There is an RL agent which learns from the consequences of its actions rather than being explicitly taught. In addition, its actions are determined by its past experiences (exploitation) and also by new choices made (exploration) [55]. The reinforcing signal that the RL agent receives is called a reward which encodes how good the outcome of an executed action was, and the agent learns how to select actions that increase its rewards accumulated over time.

In this work, the machine learning approach used for the experiments is supervised learning.

## 3.2 ARTIFICIAL NEURAL NETWORKS

In machine learning, artificial neural networks (ANNs) are machines inspired by how the brain performs certain tasks or functions [54].

The human brain has about 10 billion neurons. These structures connect to each other through synapses composing a neural network. Neurons are the most important structures of the brain and spinal cord of the central nervous system (CNS) and also of the ganglia of the peripheral nervous system (PNS) [8].

The main components of neuron, a simplified neural model generally used to build ANNs, are:

- Dendrites, which receive stimuli from other neurons;
- The cell body, which stores and combines information coming from other neurons;
- And the axon, which is shaped like tubular fiber and transmits stimuli to other nerve cells.

as shown in Figure 3.1.

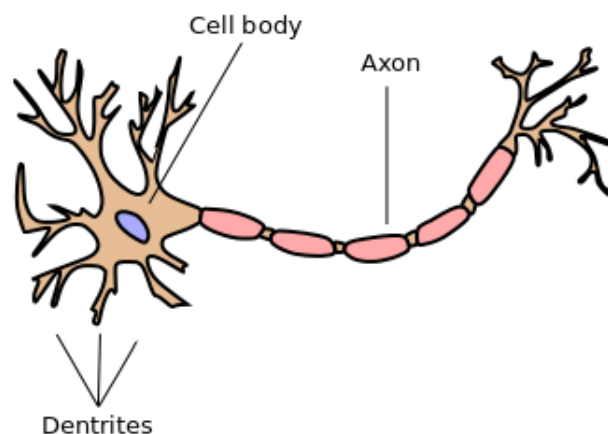


Figure 3.1: Anatomy of a neuron.

The first mentioned information about Neural computation dates back to 1943, in papers by McCulloch and Pitts [7], in which they suggested the construction of a machine based on the human brain. Hence we had our first model of computational cells known as "neurons" (as shown in Figure 3.2). The artificial neural networks generally present an interconnection between those cells which communicate with each other, so that such neurons have numerical weights,  $w_1, w_2, \dots, w_m$ , associated with them, input data,  $x_1, x_2, \dots, x_m$ , and they can be tuned according to the system experience, allowing the network to adapt to input data and be able to learn from them.

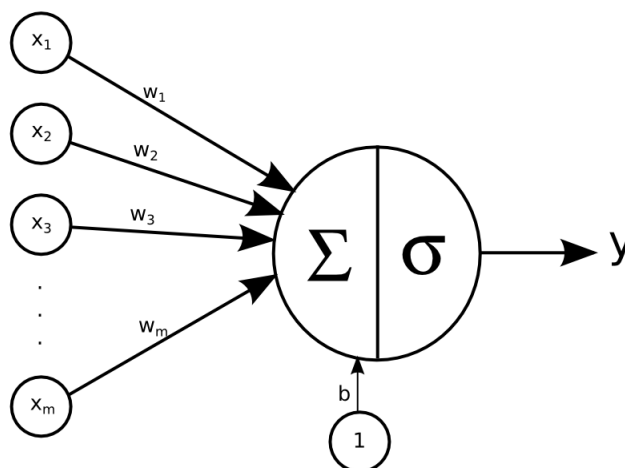


Figure 3.2: Computational model of a neuron by McCulloch and Pitts [7].

According to Haykin [8]:

*"A neural network is a massively parallel distributed processor made up of simple processing units that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:*

1. *Knowledge is acquired by the network from its environment through a learning process.*
2. *Interneuron connection strengths, known as synaptic weights, are used to store the acquired knowledge."*

For example, a neural network for handwritten digit recognition could have a set of input neurons that can be activated by the pixel values of an input image. As already mentioned, each neuron has weights associated with it, which are modified during the learning process, and the outputs of these neurons are passed on to other neurons until a neuron from the network output is activated. This activation indicates the handwritten digit that was read by the system.

Artificial neural networks have been applied, along with other machine learning methods, to a wide range of tasks, such as computer vision and speech recognition applications that would be more complicated to solve by simple rule-based programming [54]. These networks can be implemented by means of electronic components or software on digital computers. The artificial neural nets were implemented by software in this work.

### 3.2.1 Perceptron

The previously mentioned McCulloch and Pitts model inspired some later work until the first neuro-computer began to succeed in the years 1957 and 1958 [8]. This neuro-computer, called *perceptron*, was created by Frank Rosenblatt [56], Charles Wightman and others.

The *perceptron* is a mathematical model of a neuron and it works as a binary classifier. When it is combined with other counterparts, it can be used for multi-class classification in pattern recognition [56]. As shown in the model of Figure 3.3, a *perceptron* receives an input vector  $\mathbf{x} = [x_1, x_2, \dots, x_m]$  and has a weight vector  $\mathbf{w} = [w_1, w_2, \dots, w_m]$  associated to it, in which  $b$  in the Figure 3.3 is a bias, a sort of neuron threshold. In order to obtain the *perceptron* output, the dot product between the vectors  $\mathbf{x}$  and  $\mathbf{w}$  is calculated and added to  $b$  according to Equation 3.1, producing  $s$  (called induced local field), so the result is used as input of an activation function  $\sigma$  (step function in Equation 3.2) which will give us the neuron return value.

$$s = \sum_{i=1}^m x_i w_i + b \quad (3.1)$$

$$\sigma(s) = \begin{cases} -1 & , \text{if } s < 0 \\ 1 & , \text{if } s \geq 0 \end{cases} \quad (3.2)$$

$$y = \sigma(s) \quad (3.3)$$

For example, to explain the meaning of the value of  $y$ , let us consider that there are only two classes that we want to separate. In *perceptron*, when we get a value of  $s$  greater than or equal to 0,  $y$  is equal to 1 and that indicates that the input data belongs to a class  $\zeta_1$ , whereas if  $s$  is less than 0,  $y$  is equal to -1 and that gives us the input data belonging to the other class  $\zeta_2$ , as seen in Equation 3.2.

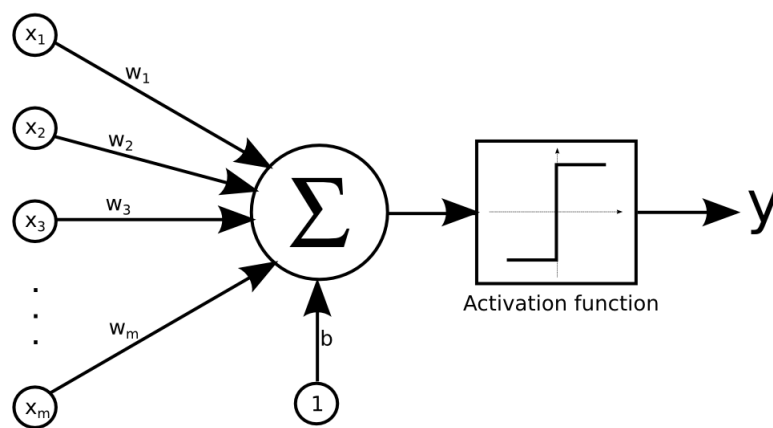


Figure 3.3: Perceptron Model

Each *perceptron* has its own weights vector and its bias  $b$ . They are very important in the

*perceptron* learning process, because these parameters are necessary to minimize a function that represents the error produced by the neuron through the search for the optimum combination of weights and bias (See Equation 3.4)

$$\varepsilon(\mathbf{w}, b) = \frac{1}{2} \sum_{j=1}^n (\sigma(\sum_{i=1}^m (w_i x_i^j) + b) - d_j)^2. \quad (3.4)$$

In the above equation, we have:

- $\varepsilon$  is the **error function**, also called **loss function**;
- $m$  is the number de elements in the weights vector;
- $n$  is the number of samples in the training set;
- $\mathbf{w}$  is the weights vector associated with the *perceptron* and  $w_i$  is the  $i$ -th element in this vector;
- $b$  is the bias of the *perceptron*;
- $\sigma$  is the activation function of the *perceptron*;
- $x_i^j$  is the  $i$ -th position of the input vector  $\mathbf{x}$  from the  $j$ -th sample in the training set;
- $d_j$  is the *perceptron* output we expect for, when it receives the  $j$ -th training sample as input.

The *perceptron* adjustable weights allow the model to learn. A numerical model is reached from supervised training. The  $\eta$  variable is the learning rate applied in the each iteration, which corresponds to the displacement step used to minimize the error  $e$  (See in Algorithm 1). If it is too large, this might make it difficult to reach the minimum value, whereas if it is too small, it might slow down convergence process. This  $\eta$  value can be fixed throughout the training, being only a value  $0 < \eta < 1$ , thus we have a *fixed-increment adaptation rule* for the *perceptron* [8]. The *perceptron* convergence algorithm is stated in Algorithm 1 and explained in details by Haykin [8]. Convergence will be achieved when all training samples have the expected output.

### 3.2.2 Multilayer Perceptron

As already stated, the simple *perceptron* is capable of separating data, as long as they are linearly separable. It can delimit a hyperplane in which the data belonging to a class are on one side, whereas the data belonging to another class is on the other side. The multilayer *perceptron* (MLP), however, allows the separation of classes of data that are not linearly separable [8].

In theory, *perceptrons* can be combined according to our desire, but in more than thirty years of research, rules for feed-forward networks have been commonly adopted in order to efficiently solve problems in pattern recognition.

---

**Algorithm 1** Perceptron Convergence Algorithm

---

**Require:**  $0 < \eta < 1$

Initialize  $\mathbf{w}(0) = \mathbf{0}$

**while** *perceptron* does not converge **do**

    Randomly select a sample  $\mathbf{x}(j)$

    Calculate  $s$  (see Equation 3.1) with sample  $\mathbf{x}(j)$

**if**  $\sigma(s)$  (see Equation 3.2) is not equal to  $d_j$  **then**

$e \leftarrow d_j - \sigma(s)$ , in which  $e$  is the error

$\Delta \mathbf{w}(j) \leftarrow \eta \cdot e \cdot \mathbf{x}(j)$

        Weights update:

$\mathbf{w}(j + 1) \leftarrow \mathbf{w}(j) + \Delta \mathbf{w}(j)$

**end if**

**end while**

---

A MLP must have one or more input neurons, that is, neurons that receive the input data from the neural network as input; A MLP must have as many output neurons as the number of classes, each of which represents a class; Given an input pattern, a MLP must associate such a pattern with the class represented by the output neurons which has the highest value among all; Normally, all neurons that belong to the same layer must share the same activation function in a MLP and the connections between them should not be cyclic.

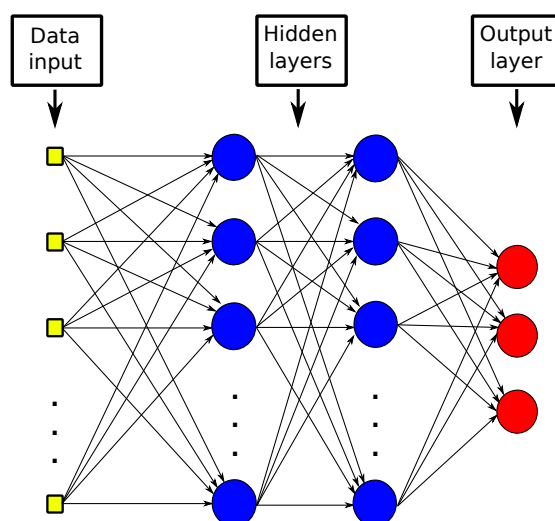


Figure 3.4: Perceptron Multilayer Example

Figure 3.4 shows a widely used MLP architecture. It is a three layer architecture, consisting of a data input, two hidden layers and one output layer. The data input is composed of propagators that feed all the neurons of the first hidden layer with some data. In the hidden layer, each neuron is connected to another in the next layer, and so it goes to the output layer that does not have output connections.

In the example in Figure 3.4, we have an artificial feed-forward neural network in which

neurons receive their input data only from previous layers and send their outputs to later layers.

In order to train a neural network, a powerful technique known as back-propagation is used [57]. With this algorithm, it is possible to adjust the weights of the entire network by propagating the error of its output backwards, that is, starting from the output layer. This technique will be explained in the next subsection.

### 3.2.3 Activation functions

In the simple perceptron, the activation function used is the step function, but to train a MLP network, using the back-propagation technique, the activation functions of the neurons must be differentiable. The commonly used activation functions are the logistic sigmoid function:

$$\text{sigm}(s) = \frac{1}{1 + e^{-s}} \quad (3.5)$$

and the hyperbolic tangent sigmoid function:

$$\gamma \tanh(\beta s) = \gamma \frac{e^{\beta s} - e^{-\beta s}}{e^{\beta s} + e^{-\beta s}}. \quad (3.6)$$

Sigmoidal functions are monotonically increasing (as illustrated in Figure 3.5) and asymptotic with smooth derivatives. The logistic function varies between 0 and 1 and the hyperbolic function varies between -1 and 1 when  $\gamma = 1$  and  $\beta = 1$ , but published results suggest to use  $\gamma = 1.7159$  and  $\beta = 2/3$  [58], because, since tanh is sometimes computationally expensive, an approximation of it by a ratio of polynomials can be used.

Sigmoidal functions that are symmetrical in relation to the origin as the hyperbolic are preferable, because their outputs tend to average to zero, which makes the input to the next layer unbiased [59], in the same way as the normalization of the network inputs make the convergence of the network faster.

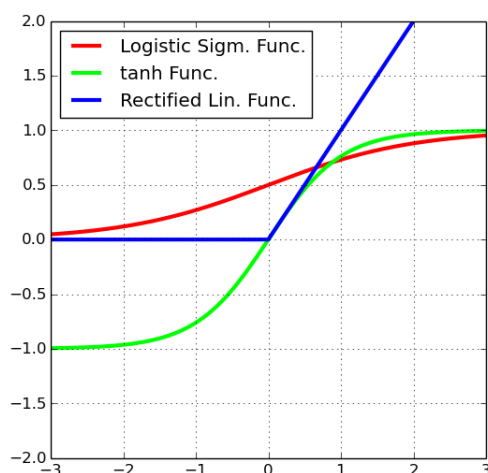


Figure 3.5: Activation functions: Logistic Sigmoid Function (in red), Hyperbolic Tangent Sigmoid Function (in green) with  $\gamma = 1$  e  $\beta = 1$  and Rectified Linear Function (in blue).

We also have the Rectified Linear Unity (ReLU) or rectified linear function of equation:

$$relu(s) = \max(0, s) \quad (3.7)$$

and shown in Figure 3.5. This activation function has been widely used in the context of convolutional neural networks, since it has shown an improvement in the discriminative capacity of neural networks [60]. Tests performed by Krizhevsky in [23] showed convergence 6 times faster using ReLU activations when compared to the hyperbolic tangent in equivalent networks.

Usually for classification problems, the last layer of the network uses another activation function, called **Softmax**. It is useful for the last layer of the network mainly by allowing a probabilistic interpretation over network outputs. The **Softmax** function is defined by:

$$y_i^l = \frac{e^{s_i^l}}{\sum_j e^{s_j^l}}, \quad (3.8)$$

where  $s_i^l$  is the induced local field of the neuron  $i$  and  $y_i^l$  is the output of the neuron  $i$  after passing through the Softmax function, which divides the exponential of the induced local field from neuron  $i$  by the sum of the exponentials of the induced local fields from all the neurons in the layer.

### 3.2.4 Error Functions

The error function, also known as loss function, is defined in order to train an artificial neural network. This function calculates the error of the network predictions made on the dataset. In the training process, we wish to minimize the sum of this error function over all samples in the dataset.

The commonly used loss functions are the square error function (SEF), described by Equation 3.9, and the cross-entropy error function (CEF), described by Equation 3.10, in which both calculate the error for a single sample of the dataset.

In these equations  $d_j$  and  $y_j^l$  respectively represent the desired output of neuron  $j$  and the output obtained by neuron  $j$  from layer  $l$ . Entropy requires positive values of  $y_j^l$ .

$$\varepsilon = \frac{1}{2} \sum_j (y_j^l - d_j)^2 \quad (3.9)$$

$$\varepsilon = - \sum_j [d_j \log(y_j^l) + (1 - d_j) \log(1 - y_j^l)] \quad (3.10)$$

It can be shown that the true posterior probability is a global minimum for both the cross-entropy function (CEF) and squared error function (SEF). Thus, in theory a neural network can be trained equally by minimizing either function. In practice, however, CEF leads to faster convergence and better results in terms of classification error rates, which made it to become more popular in recent years [61].



### 3.2.5 Optimizers

The training of neural networks aims to modify the parameters of the model, so that the output layer produces the desired results. As already mentioned, learning the parameters requires the minimization of the loss function. In order to do it, we can use Stochastic Gradient Descent (SGD) [62], stated in Equation 3.11:

$$w_{ji}^l(n+1) = w_{ji}^l(n) - \eta \frac{\partial \varepsilon}{\partial w_{ji}^l(n)}, \quad (3.11)$$

where  $\eta$  is the learning rate,  $w^l(n)$  are the weights of the layer  $l$  at iteration  $n$  and  $\varepsilon$  is the loss function.

In general, to estimate the direction of steepest gradient descent an average gradient over subset (called batch) of training examples is used instead of the complete training set. Batch size is usually determined empirically. It regulates the trade-off between the variance of gradient estimative and computational time.

Determining a good learning rate  $\eta$  makes optimization better. If  $\eta$  is too high, the network may not reach the desired local minimum, if  $\eta$  is too low, the learning process can be very slow. In order to alleviate this process, we might use ADADELTA [63] optimizer, which adapts the learning rate during training process. At iteration  $n$  ADADELTA updates the parameters as stated in Equation 3.12 [63].

$$w_{ji}^l(n+1) = w_{ji}^l(n) - \frac{RMS[\Delta w_{ji}^l]_{n-1}}{RMS[g]_n} g_n, \quad (3.12)$$

where

$$RMS[g]_n = \sqrt{E[g^2]_n + \epsilon} \quad (3.13)$$

and

$$E[g^2]_n = \rho E[g^2]_{n-1} + (1 - \rho) g_n^2. \quad (3.14)$$

Equation 3.14 is an exponentially decaying average of the squared gradients  $g^2$  with a decay rate  $\rho$ . Similarly,  $RMS[\Delta w_{ji}^l]_{n-1} = \sqrt{E[(\Delta w_{ji}^l)^2]_{n-1} + \epsilon}$  is an average of the decay of previous updates and  $\epsilon$  is a small constant that ensures the existence condition of the  $RMS[g]_n$  denominator and the beginning of the first iteration, when  $E[g^2]_0 = 0$ . ADADELTA assigns each dimension in the space of parameters its own dynamic learning rate with some desired properties, which are detailed in Zeiler's paper [63].

### 3.2.6 The Back-propagation Algorithm

In the back-propagation algorithm, at each iteration, the network will be trained with all input patterns of the training set. Initially, the vectors of initial weights of each neuron should be generated randomly. If they were initialized with the same value, all neurons in each layer would be updated with the same value, which would cause the entire system to learn only one feature.

Then, the first layer of the network is fed with the input signals  $\mathbf{x}$  and the forward propagation of the signals is processed. It is processed as in the simple *perceptron*, where, for the neuron  $j$  in layer  $l$ , the local induced field  $s$  in iteration  $n$  is:

$$s_j^l(n) = \sum_i w_{ji}^l(n) y_i^{l-1}(n), \quad (3.15)$$

where  $y_i^{l-1}(n)$  is the output of the previous layer. Then, applying the activation function  $\sigma$  on the induced local field  $s_j^l(n)$ , we obtain the neuron output of the layer  $l$ , given by:

$$y_j^l(n) = \sigma_j(s_j^l(n)). \quad (3.16)$$

If the neuron belongs to the first hidden layer, its output will be:

$$y_j^1(n) = \sigma_j\left(\sum_i w_{ji}^1(n) x_i(n)\right), \quad (3.17)$$

where  $x_i(n)$  is the  $i$ -th position of the input vector  $\mathbf{x}$  in the  $n$ -th iteration.

If the neuron  $j$  belongs to the output layer, that is,  $L$  being the depth of the network and  $l = L$ , we have:

$$y_j^L(n) = o_j(n), \quad (3.18)$$

where  $o_j$  is the output obtained by neuron  $j$  from the output layer. Thus, the calculated error in the last layer is given by:

$$e_j(n) = d_j(n) - o_j(n), \quad (3.19)$$

where  $d_j(n)$  is the expected value for the output of neuron  $j$  from the last layer.

Finally, the back-propagation algorithm, in which the error is propagated backwards through the layers, calculating the local gradients of the error function with respect to the weights,  $\delta_j^l$ , as can be seen in Equation 3.20:

$$\delta_j^l(n) = \begin{cases} e_j^L \sigma_j'(s_j^L(n)), & \text{for the neuron } j \text{ in the output layer } L \\ \sigma_j'(s_j^l(n)) \sum_k \delta_k^{l+1}(n) w_{kj}^{l+1}(n), & \text{for the neuron } j \text{ in a hidden layer } l \end{cases} \quad (3.20)$$

where  $\sigma'$  represents the derivative of the neuron activation function.

After calculating the error, the weights of the neurons are adjusted by means of the Delta Rule [8] shown in the following equation:

$$w_{ji}^l(n+1) = w_{ji}^l(n) + \alpha[\Delta w_{ji}^l(n-1)] + \eta \delta_j^l(n) y_i^{l-1}(n), \quad (3.21)$$

with  $\eta$  being the learning rate and  $\alpha$  the *momentum* constant, a sort of penalty on the weights of the previous iteration to update the weights of the future iteration.

The forward propagation of information and the backward propagation of the error will be repeated until the stopping criterion used has been fulfilled.

### 3.2.7 Generalization

In back-propagation learning, we use a training sample to compute the synaptic weights of a multilayer *perceptron*. The network is trained by loading as many of the training sample examples as possible and we expect the neural network designed to generalize well. A network is said to generalize well when the input–output mapping computed by the network is correct (or nearly so) for test data never used in creating or training the network [8].

The learning process of a neural network can be seen as a fitting problem. The network can be considered as a non-linear input-output mapping. So we can consider generalization as the effect of good non-linear interpolation of input data.

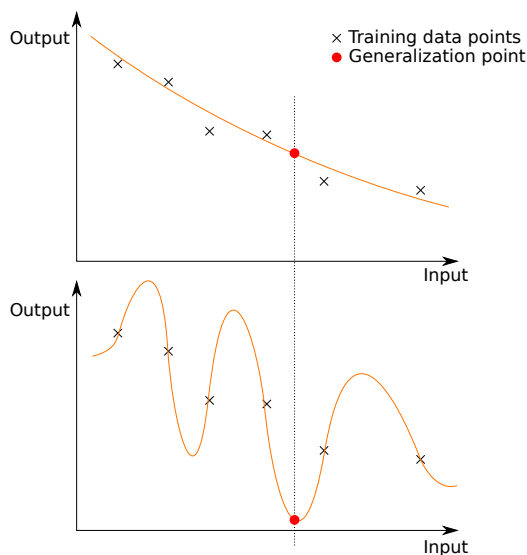


Figure 3.6: Properly fitted nonlinear mapping (upper graph) and overfitted nonlinear mapping (lower graph). Adapted from [8]

As we can see in Figure 3.6, in the upper graph the model obtained with the training data presents a nonlinear mapping properly fitted, whereas in the lower graph, the mapping is overfitted, with poor generalization, because even if it passes through the training points very well, for input data that were not used in training, it presents a large error interpolation.

A neural network that is designed to generalize well will generate a correct input-output mapping, even when the input from the network is slightly different from the examples used to train it [8], as illustrated in Figure 3.6. However, if a neural network learns from many examples, it can store (or “memorize”) the training data. It can do so due to the noise present in the training samples, which do not really characterize the function that must be modeled. To this phenomenon we call *overfitting* or *overtraining*.

### 3.3 DEEP LEARNING

Deep learning (deep machine learning, or hierarchical learning, or sometimes DL) is a part of machine learning that comprises a set of algorithms. Those algorithms model high level abstractions from training data through model architectures with complex structures. Otherwise, is composed of multiple non-linear transformations [64, 65]. Deep learning belongs to a broader family of machine learning methods that are based on the representations of learning data. An observation of an image, for example, which is presented as input to a deep learning architecture, and can be encoded in several ways: as a vector of pixel intensities, a set of horizontal, vertical or diagonal edges (in a more abstract way), gradients, specific shapes found in these images. Some representations found in these architectures allow the system to learn tasks or classify objects more easily (for example, recognition of facial expressions) from examples [66].

The most representative characteristic of deep learning is the depth of their networks. In theory, a two layer model has universal approximation property, but in practice multiple layers proved essential. With the increasing amount of data and computational power now available, the construction of deep neural networks with a large number of layers has been advantageously possible. Another important feature about deep learning is the ability to replace handcrafted features with efficient algorithms for learning unsupervised or semi-supervised features as well as extracting hierarchical features [67].

Some of the DL representations are inspired by advances in neuroscience and are based on the vague interpretation of information processing and communication patterns of the nervous system, such as neural coding that attempts to define a relationship between stimuli and neuronal responses in the brain [68].

Many deep learning architectures such as deep neural networks, deep convolutional neural networks, recurrent neural networks and deep belief networks have been applied in fields such as natural language processing, speech recognition, computer vision and bioinformatics, showing good results in the state-of-the-art [69].

#### 3.3.1 CNN: Convolutional Neural Networks

Convolutional networks are based on three main ideas: local receptive fields, shared weights and spatial or temporal sub-sampling [10]. This type of neural network has been widely used for object or face recognition problems, and has recently obtained results in more challenging tasks such as the ImageNet Large Scale Visual Recognition Challenge [70]. In their structure, they receive as input an image. Typically, the training process of these networks is accelerated by the use of GPUs (Graphical Processing Units). In the following subsections, we will describe the types of layers of convolutional neural networks.

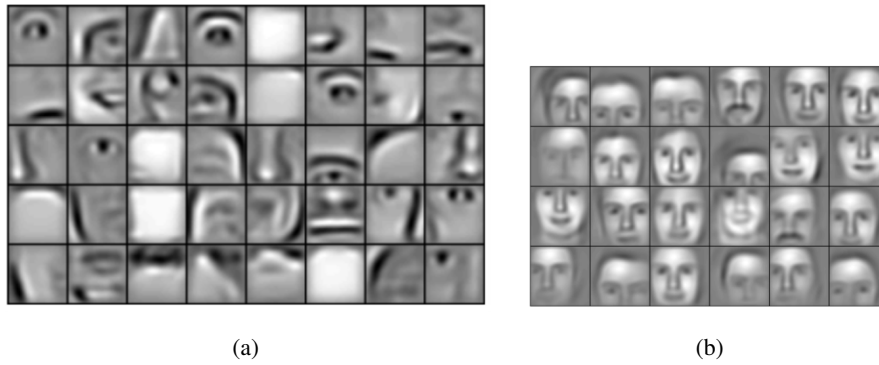


Figure 3.7: (a) represents the second layer (feature maps) (low level features) learned from faces dataset. (b) represents the third layer (feature maps) (higher level features) learned from faces dataset. Both of them illustrated by Lee et al. [9].

Figure 3.7 illustrates an example of *feature maps* learned in the second and third convolution layers of a convolutional neural network, used to classify faces. As shown in Figure 3.7, lower layers have low level features (more generic ones) and deeper layers have higher level features (more specific ones). We observe that this type of neural network model can identify detectors of interesting characteristics, such as straight edges detector, simple colors (for color images) and curves.

### 3.3.1.1 Convolutional Layer

Convolutional layers have filters, also known as feature maps (see Figure 3.10), which are trainable and are applied to the entire input image [68]. Each filter is an array of neurons, in which each neuron is connected only to a subset of the neurons of the previous layer, called the *receptive field*. When the input is an image, the filters define a convolution kernel with small area, usually 3x3 or 5x5 pixels, and each map neuron is connected only to the pixels that are contained in that area of the image. If previous layer is a matrix of neurons, it is the same rule. The weights of the neurons of the same *feature map* are shared, allowing them to learn patterns that occur frequently in any part of the image.

The definition for a 2D convolution layer is given in Equation 3.22. In this equation, the discrete convolution is applied to the inputs  $y^{l-1}$  with a set of weights  $w^l$ , by adding a bias  $b^l$  and then applying the non-linear activation function  $\sigma$ :

$$y_{mn}^l = \sigma\left(\sum_{i=1}^{N_m} \sum_{j=1}^{N_n} y_{(m+i-1)(n+j-1)}^{l-1} w_{ij}^l + b^l\right), \quad (3.22)$$

where  $y_{mn}^l$  is the output neuron at position  $(m, n)$ ,  $w_{ij}^l$  is the weight value at position  $(i, j)$  of the kernel,  $N_m$  and  $N_n$  are the number of rows and columns of the 2D kernel,  $b^l$  is the bias and  $y_{(m+i-1)(n+j-1)}^{l-1}$  is the value of the input of this layer (or output of the previous layer) at position  $(m + i - 1, n + j - 1)$ .

Equation 3.22 represents the output of the neuron at position  $(m, n)$  on the *feature map*, in which  $m \in \{1, \dots, S_m - N_m + 1\}$  and  $n \in \{1, \dots, S_n - N_n + 1\}$ , where  $S_m$  and  $S_n$  are the number of rows and columns of the input matrix of this layer  $l$ . But the convolution layer can be applied to all possible inputs or not. It is also possible to apply the convolution only to inputs that are spaced apart by a distance  $s$ , known as *stride*, which can be useful, for example, to reduce computational time and reduce area overlapping between *receptive fields*. In Figure 3.8, we have, as an example, the forward propagation of the convolution layer with stride  $s = 1$ .

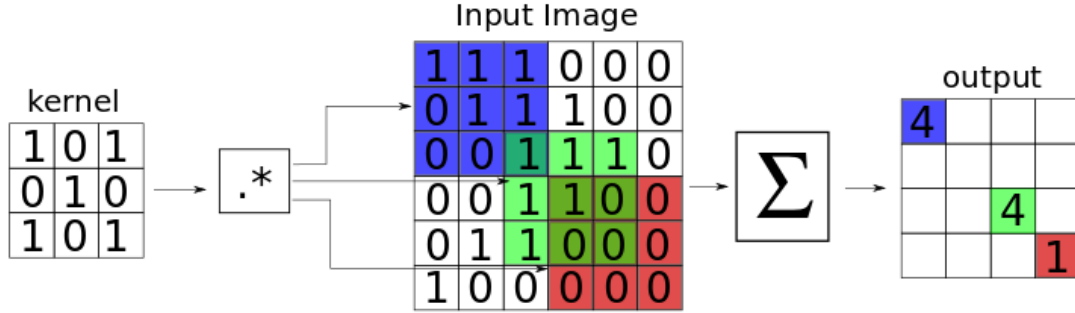


Figure 3.8: The forward propagation of a convolutional layer.

In convolutional layers all the weights are learned with back-propagation and convolutional networks can be seen as synthesizing their own feature extractor, according to LeCun *et al.* [10].

### 3.3.1.2 Pooling Layer

Pooling layers apply a non-linear downsampling function on their previous layer in order to reduce their size and collect information from invariance to translation and other distortions. There are two forms of pooling layer, known as average pooling and max pooling. In this work we will use the max pooling layer, whose function is formulated by:

$$y_{mn}^l = \max_{i,j \in \{0,1,2,\dots,p\}} y_{(m+i-1)(n+j-1)}^{l-1}, \quad (3.23)$$

where  $y_{mn}^l$  is the output at position  $(m, n)$ ,  $y_{(m+i-1)(n+j-1)}^{l-1}$  is the input value at position  $(m + i - 1, n + j - 1)$  and  $p$  is the size of the area in which the max pooling function will be applied. Similarly to the convolution layer, the pooling layer also has a *stride* parameter  $s$ . In Figure 3.9, we illustrate the forward propagation phase of pooling with  $s = 1$ , that is, passing through all possible pooling windows.

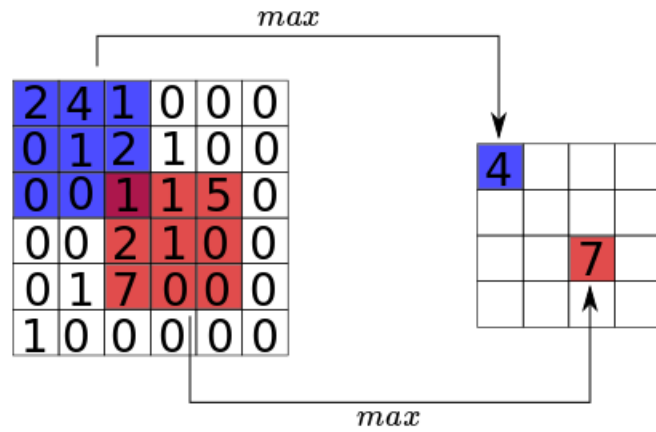


Figure 3.9: The forward propagation of a convolutional layer.

Pooling layers make the CNN architecture more robust, providing a certain degree of invariance to translation because the activated neurons are independent of the spatial location of the image features within the pooling window [71]. According to Scherer *et al.* [72], max pooling layers presented better results than other pooling architectures.

Figure 3.10 shows an example of a convolutional network model.

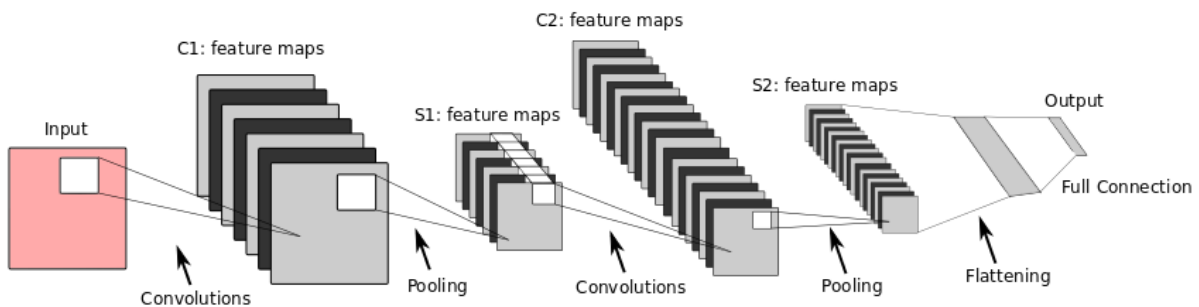


Figure 3.10: An overview of a Convolutional Neural Network. Adapted from LeCun et al. [10]

### 3.3.1.3 Dropout

Dropout is a technique used to prevent neural networks from overfitting and to provide a way of combining exponentially many different neural network architectures efficiently [73]. During the training process of a neural network, each neuron has probability  $p$  to be temporarily removed from the network, as shown in Figure 3.11. This is therefore done randomly. This neuron with dropout probability  $p$  will be ignored both in the forward pass and in the back-propagation process during training. By dropping out a unit, we mean that we temporarily remove the unit from the network along with all its input and output connections. This technique prevents neurons from co-adapting too much.

For example, if a layer has  $n$  neurons, we have  $2^n$  different possible neural networks during

the training process. These networks share weights and for each presentation of each training sample, a new network is sampled and trained. But when the model is tested, no neurons are dropped out and their weights will be scaled by  $p$ , which makes it possible for  $2^n$  networks with the same parameters to be combined in one neural network.

The weights of this combined network are scaled-down versions of the trained weights from each one of the  $2^n$  networks. If a neuron is retained with probability  $p$  during training, the outgoing weights of that neuron are multiplied by  $p$  during test process.

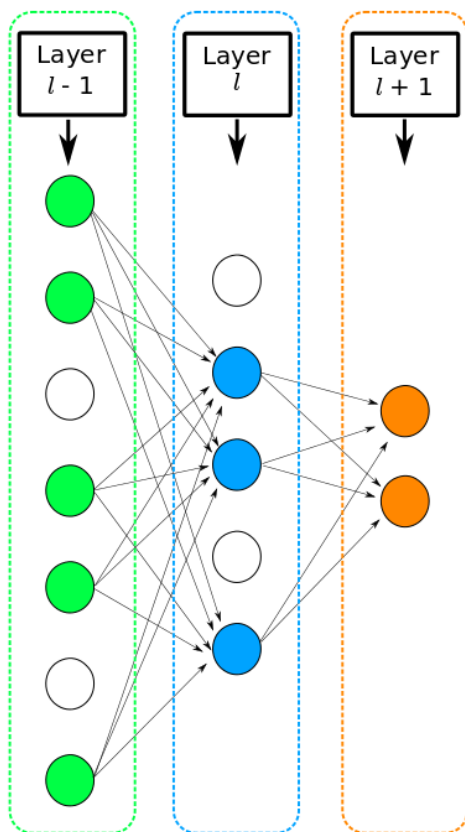


Figure 3.11: Dropout illustration. Neurons in white color are dropped out.

### 3.3.1.4 L1 and L2 Regularizers

Regularization is a selection of complexity level or tuning of the model that allows the classifier model to have a better performance in its classification. A central problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs [74], and regularization helps to achieve this goal.

Regularization is necessary especially when the data set is of a high dimensionality and does not have many data instances. In the case of artificial neural networks, which is the focus of this work, the loss function can be regularized by adding to it a penalty function:



Table 3.1: Main differences between L1 and L2 regularization

<b>L1 Regularization</b>	<b>L2 Regularization</b>
Computational inefficient on non-sparse cases	Computational efficient (analytical solutions)
Sparse outputs	Non-sparse outputs
Built-in feature selection	No feature selection

$$\varepsilon_R = \varepsilon + R(\mathbf{w}), \quad (3.24)$$

where  $\varepsilon$  is the loss function indicated in Equations 3.9 and 3.10,  $\varepsilon_R$  is the regularized loss function,  $\mathbf{w}$  is the weight array of the neural network and  $R(\mathbf{w})$  is the regularizing penalty function.

There are two types of regularization functions most used, they are L1 and L2 regularization, given by:

$$L1 : R(\mathbf{w}) = \lambda \sum_{i=1}^n |w_i| \quad (3.25)$$

and

$$L2 : R(\mathbf{w}) = \lambda \sum_{i=1}^n w_i^2, \quad (3.26)$$

where  $n$  is the number of components of the array  $\mathbf{w}$ ,  $w_i$  is the  $i$ -th component of the array  $\mathbf{w}$  and  $\lambda$  is the regularization parameter.

Conceptually, L1 regularization, which is known to induce sparsity, requires a number of samples proportional to the logarithm of irrelevant features and L2 regularization, also known as Euclidian norm, requires a number of samples proportional to irrelevant features. We show L1 and L2 regularization main differences in Table 3.1 and, mathematically, their Equations 3.25 and 3.26.

By applying the penalty function  $R(\mathbf{w})$  on the loss function, when the gradient is applied, this will cause a natural tendency for weights that are not essential for network performance to be decreased.

### 3.3.2 Transfer Learning

In general, to train convolutional neural networks from scratch, it is necessary a dataset of sufficient size because with so many parameters and few training samples, it can easily generate overfitting. So it has been very common to pre-train a CNN in a very large dataset, such as ImageNet [67], which has 1.2 million images with 1000 categories. Then we use such a network just for training initialization or as a feature extractor for another classifier. This practice is known as Transfer Learning, which can be used in two main ways [75]:

- **CNN as a feature extractor:** We take a CNN pre-trained (VGG16 [39] network) on a very

large dataset, such as ImageNet, then we remove the last fully-connected layer, that is, the layer with outputs which are the class scores. We treat the rest of the network as a feature extractor for our new dataset and train a multilayer perceptron (MLP) with (**Softmax**) on these features;

- **CNN Fine-tuning:** In the second strategy, besides training a non-linear classifier on the new dataset using the pre-trained CNN as a feature extractor, we fine-tune the weights of the pre-trained network by back-propagation. We do not train some of the layers, in order to avoid overfitting, and only train some layers of the network, because lower layers contain more generic features and higher ones, more specific features to the details of the classes from the original dataset used to pre-train the network [75].

The two main factors taken into account when deciding how to do fine tuning are the size of the new dataset (small or large) and its level of similarity to the dataset originally used to pre-train the network. Remembering that generic features are in shallower layers of the convolutional networks and in deeper layers, we have features more specific to the original dataset, we have the following rules for fine-tuning:

- *New dataset is small and similar to the original dataset:* Because the dataset is small it is not interesting to perform fine-tuning because it can generate overfitting. As the new dataset is similar to the original, we expect high-level features of the network to be beneficial to the new dataset, so we train a classifier on the features generated from the high-level layers of the pre-trained network.
- *New dataset is large and similar to the original dataset:* When the dataset is really large, we do not worry about overfitting and because of its similarity as the original dataset, we can fine-tune across the entire network.
- *New dataset is small and very different from original:* When the dataset is small and very different from the original dataset it is better to train a classifier from features generated in lower layers, since the higher ones would have more specific information from the original dataset.
- *New dataset is large and very different from the original:* Because the dataset is large, we could simply train CNN from scratch. However, we can benefit from the weights already obtained by the pre-trained network. In this case, we would have enough confidence to fine-tune through the entire neural network.

A smaller learning rate is commonly used for CNNs weights that are being fine-tuned, when compared to the weights randomly-initialized for the new classifier added on top of the network to train the new dataset. This is because we do not want to distort the CNN weights too quickly, since they are relatively good based on the CNN pre-training [75].

The VGG16 is a CNN [39] pre-trained on the largest and most challenging ILSVRC-2014 dataset, which have achieved good accuracy rates in object recognition. We will use it in our methodology. Its architecture is described in Table 3.2.

ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories. The images were collected from the web and labeled by human labelers using Amazon’s Mechanical Turk crowd-sourcing tool. Starting in 2010, as part of the Pascal Visual Object Challenge, an annual competition called the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) has been held. ILSVRC uses a subset of ImageNet with roughly 1000 images in each of 1000 categories. In all, there are roughly 1.2 million training images, 50,000 validation images, and 150,000 testing images [23]. This dataset have in its categories different types of animals, plants, sport activities, material (fabric), instrumentation and tools, scenes and foods.

Table 3.2: VGG16 Architecture. C denotes a Convolution layer, MP denotes a max pooling layer and FC denotes a fully-connected layer.

Layer	Type	Output Maps (neurons)	Kernel size	Pooling size
1	C	64 feature maps	3x3	-
2	C	64 feature maps	3x3	-
3	MP	64 feature maps	-	2x2
4	C	128 feature maps	3x3	-
5	C	128 feature maps	3x3	-
6	MP	128 feature maps	-	2x2
7	C	256 feature maps	3x3	-
8	C	256 feature maps	3x3	-
9	C	256 feature maps	3x3	-
10	MP	256 feature maps	-	2x2
11	C	512 feature maps	3x3	-
12	C	512 feature maps	3x3	-
13	C	512 feature maps	3x3	-
14	MP	512 feature maps	-	2x2
15	C	512 feature maps	3x3	-
16	C	512 feature maps	3x3	-
17	C	512 feature maps	3x3	-
18	MP	512 feature maps	-	2x2
19	FC	4096 neurons	-	-
20	FC	4096 neurons	-	-
21	FC	1000 neurons	-	-

### **3.4 FINAL CONSIDERATIONS**

Artificial Neural Networks (ANNs) are robust tools commonly used to solve problems that require learning, such as pattern detection and recognition. In this Chapter we approached a vision about Machine Learning and described the main concepts of Artificial Neural Networks, showing their main algorithms. Within the field of Deep Learning research we approached theories about Convolutional Neural Networks (CNNs), which in turn were used to classify human actions in this work.

In the following Chapter we will present the proposed methodology for the human action recognition in image sequences, in which we use stacks of Dense Optical Flow (DOF) applied to a temporal convolutional network (Temporal stream) and apply raw images to another convolutional network, called Spatial stream.

## 4 METHODOLOGY

In order to classify the type of human action performed in a sequence of images, we propose the methodology illustrated in Figure 4.1, which uses certain techniques of Computer Vision and Computational Intelligence combined. In the Computer Vision stage, we calculate the Dense Optical Flow (DOF) of the motion presented in the sequence of images and generate stacks of Optical Flow of the same size from each sequence. These stacks are applied to a convolutional neural network, here called the Temporal CNN stream, during its training process. We extract the frames from the sequence of images that feed another convolutional neural network, here called Spatial CNN stream. The spatial network is trained by using the transfer learning technique applied to the pre-trained VGG16 convolutional network [39].

In different moments of the video, the Spatial CNN stream may indicate different classes based on the frames of the sequence. By hypothesizing that certain static frames can lead to a correct classification, we generate a histogram to make a final classification based on the most frequent class assignment. Then, after training both convolutional networks, we will have the Spatial CNN stream histogram and, in the case of Temporal CNN stream, we also generated a histogram showing the number of stacks of Optical Flow classified in each class of the problem. After generating the histograms, they are normalized and summed, so that the resulting histogram will classify the action of the sequence of images based on the class with the highest value. At different moments in the video the system can indicate different classes, and so we generate a histogram to do the final classification based on the assignment of the most frequent class. In fact, we are adopting a somewhat bold hypothesis: that certain static frames can lead to correct classification of actions, as well as stacks of Optical Flow generated from the video sequence.

In the following sections we describe the inputs and outputs of each stage of the methodology, as well as the parameters used to configure the Computer Vision and Computational Intelligence algorithms.

### 4.1 APPROACH

In order to classify the action performed in a sequence of images, we propose the methodology illustrated in Figure 4.1.

This approach is based on Convolutional Neural Networks [10] and in this proposed methodology two streams are used: The first one as a Spatial Stream and the second one as a Temporal Stream. In this paper, an architecture based on the Lenet [10] architecture, originally created for image recognition, was used to recognize actions in videos.

Figure 4.1 shows a sequence of images as the input of the system. This sequence is input to

the spatial network (Spatial CNN), each frame of the sequence is evaluated by the network and, afterwards, we generate a normalized histogram of classification of these frames, which presents the frequency of each class as the sequence passes through the spatial network. In parallel, the sequence of input images of the system goes through a step of calculating the optical flow between its consecutive frames. Then, stacks of optical flow components are generated and will be used as temporal network (Temporal CNN) input, see Figure 4.1. In the same way as in the spatial network, we generate a normalized histogram of classification of these stacks for the temporal network. To perform the final classification of the action in the sequence, we sum these histograms and the resulting histogram gives us the sequence classification by its highest frequency class.

The Spatial stream was trained from the frames of the sequence of images and the Temporal stream was trained with stacks of Farneback Dense Optical Flow (DOF) [6], also used by Simonyan and Zisserman [3]. The spatial stream was trained by using a transfer learning technique applied to the pre-trained VGG16 convolutional network [39] (pre-trained on the largest and most challenging ILSVRC-2014 dataset, which have achieved good accuracy rates in object recognition), in order to take advantage of the features previously learned by the network.

The two networks were trained separately and for each sequence we generated a histogram. Each histogram is generated by the number of assignments to each class as the sequence of images is presented to the each stream then we combine them in order to classify the sequences based on the most frequent class assignment.

## 4.2 SPATIAL STREAM

The input samples of the spatial network were the frames of each video (sequence of images) of the training dataset. We used the pre-trained VGG16 network (see Table 3.2) in the ImageNet dataset [39], which was trained to classify 1000 categories of objects. As indicated in Figure 4.2, we did not train the first three convolutional blocks (each convolutional block is separated from the others by a MaxPooling layer) and trained the last two ones. Then, we trained a multilayer perceptron (MLP) by fine-tuning the network with the new dataset features, as we mentioned in Subsection 3.3.2. The weights of the net were learned using the Stochastic Gradient Descent (SGD) and fine-tuning should be done at a very slow learning rate, and usually with the SGD optimizer instead of an adaptive learning rate optimizer. This is to ensure that the magnitude of the updates remain very small, so as not to destroy previously learned features. If the frames of the analyzed dataset had resolution less than  $224 \times 224$ , we used their original dimensions, otherwise we set them to  $224 \times 224$ .

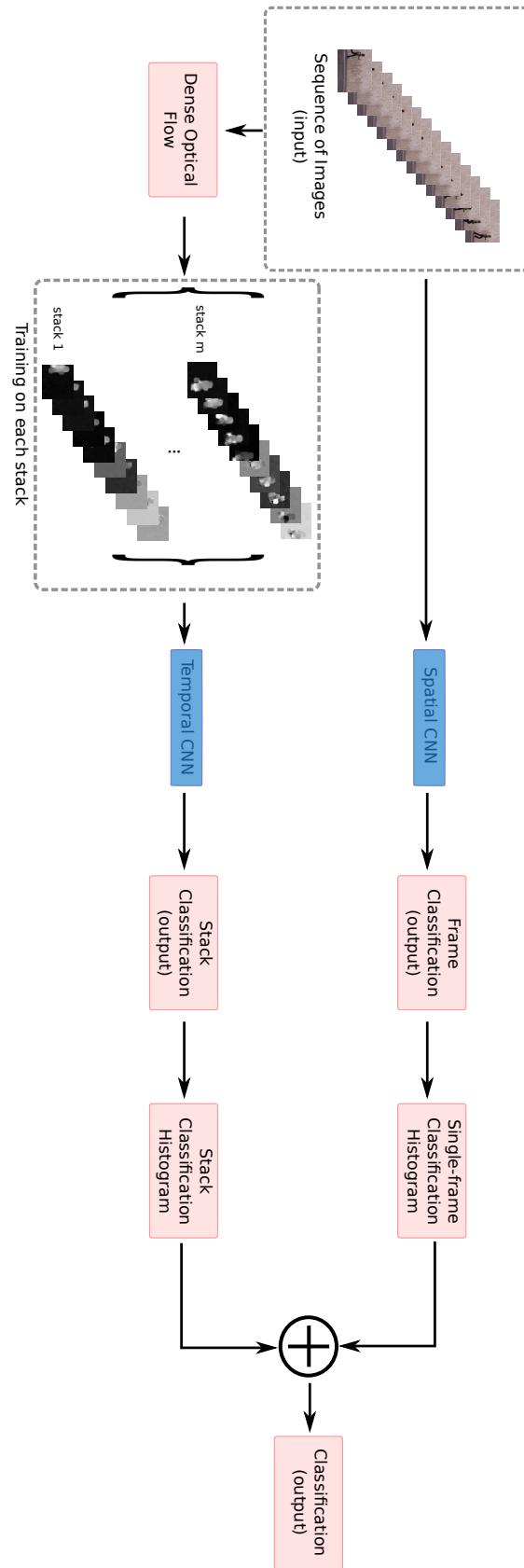


Figure 4.1: Proposed Method.

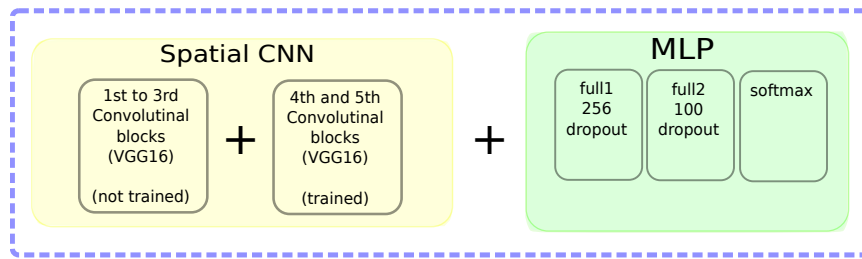


Figure 4.2: Spatial CNN structure: it refers to the Spatial CNN block in the Figure 4.1. The yellow block represents VGG16 convolutional blocks and the green block represents the MLP used. In the yellow block only the fourth and fifth blocks were trained.

### 4.2.1 Spatial Stream Histogram

Once the network is trained with the new dataset, we are able to classify each frame coming from any sequence of images. Then, to classify the action present in the sequence, we generate a histogram based on the frequency of each assigned class as the sequence is presented. Final class assignment is to the class assigned to most of the frames in the sequence. Figure 4.3 shows an example of histogram based on classes frequency.

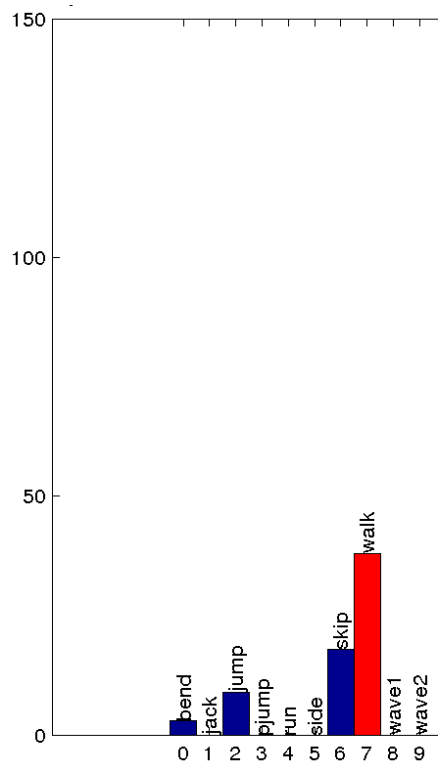


Figure 4.3: Histogram of frames classification for Sequence 8 (Weizmann Dataset). The red bar indicates the expected class and the blue bars indicate the other classes.



### 4.3 TEMPORAL STREAM

The samples used for temporal network training are stacks of Dense Optical Flow (DOF), also similarly used by Simonyan and Zisserman [3], generated from the input sequence of images (Figure 4.1). The displacement between frames is determined by finding such an estimate for two consecutive frames by using Farneback’s algorithm [6].

#### 4.3.1 Stacks of Dense Optical Flow

We calculate the optical flow of the input sequence and then apply the optical flow stacking according to Algorithm 2. In a stack of size  $D$ , we stack  $D/2$  horizontal components,  $D/2$  corresponding vertical components and then stack the resulting stacks. The structure of our temporal network is illustrated in Figure 4.4. Figure 2.9 (in page 18) illustrates an example of two consecutive frames and horizontal and vertical optical flow components respectively.

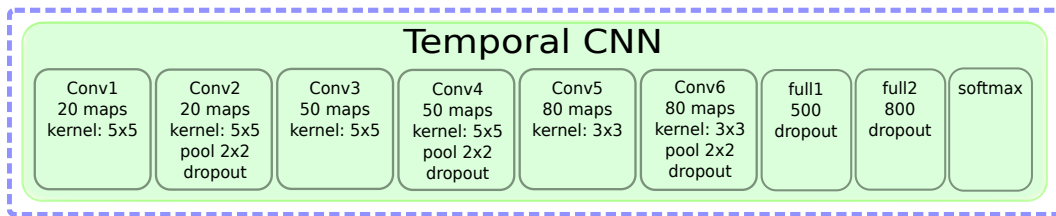


Figure 4.4: Temporal CNN structure: it refers to the Temporal CNN block in the Figure 4.1.

---

#### Algorithm 2 Stacks of Dense Optical Flow

---

**Input:** sequence of images, overlap size between stacks ( $o$ ), stack size ( $D$ )

**Output:** stacks of DOF [6] from the sequence.

Calculate horizontal ( $Dx$ ) and vertical ( $Dy$ ) DOF components between each two consecutive frames of the sequence.  $fx$  and  $fy$  are empty lists for the stacked components.

$N = \text{number of frames} - 1$

$step = (D/2) - o$

**for**  $i = 1$  **to**  $N$  **step**  $step$  **do**

**if**  $((N - i) \geq (D/2) - 1)$  **then**

**for**  $j = i$  **to**  $i + (D/2) - 1$  **do**

Append  $Dx[j]$  to  $fx$  and append  $Dy[j]$  to  $fy$

**end for**

Stack  $fx$  elements into  $flowX$  and stack  $fy$  elements into  $flowY$

$stack = \text{stacks}(flowX, flowY)$

Append  $stack$  to  $stacksList$

**end if**

**end for**

Stack  $stacksList$  elements to get the final  $output$

**return**  $output$

---

We have an illustration of stacks of Dense Optical Flow (DOF) generated from a sequence of images in the Figure 4.5.

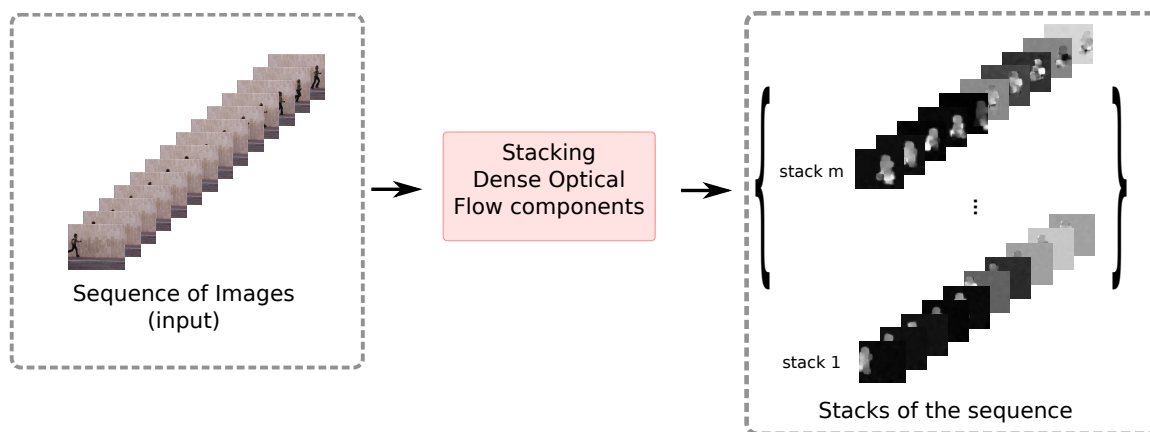


Figure 4.5: Stacks of Dense Optical Flow.

### 4.3.2 Temporal Stream Histogram

With the temporal network trained, we can classify each stack coming from any sequence of images. In order to classify the action of the sequence, we then use the same procedure for the spatial CNN. Final class assignment is the class assigned for most of the stacks. Figure 4.6 shows an example of histogram based on classes frequency.

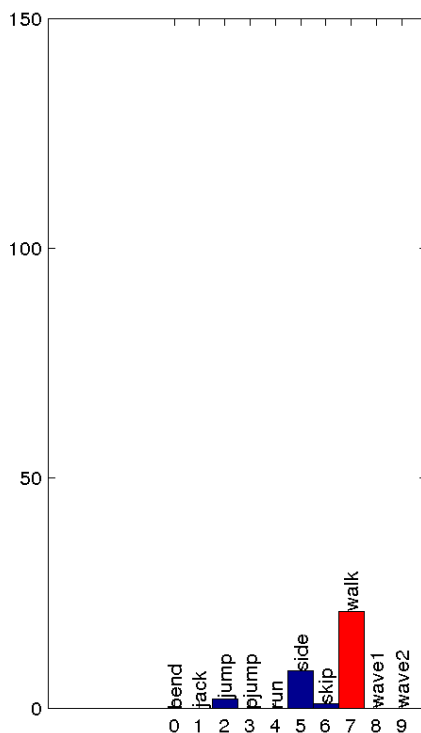


Figure 4.6: Histogram of OF stacks classification for Sequence 8 (Weizmann Dataset). The red bar indicates the expected class and the blue bars indicate the other classes.

## 4.4 COMBINATION OF CLASSIFIERS

After both networks were trained, we can have the histograms mentioned in Subsections 4.2.1 and 4.3.2, that indicate the frequency of each class for all frames (for the Spatial stream) and stacks (for the Temporal stream). We then normalize these histograms and we finally classify the videos (image sequences) by summing spatial and temporal normalized histograms and choosing the class with the higher frequency of classification in the resulting histogram.

In the next Chapter we will present the datasets used in this work, the experimental setup used, some experiments carried out and their results.

## 5 RESULTS

In this Chapter we present the results obtained with the implementation of the prototype proposed in Chapter 4, in which the techniques presented in Chapters 2 and 3 were used.

Our prototype was developed in the Python language. To calculate the Dense Optical Flow in the image sequences we use the OpenCV library [76], developed in C and C ++, with a Python version. This library can be used in Mac, Windows and Linux operating systems. For the development of Convolutional Neural Network (CNN) algorithms, we used the Keras framework [77], developed in Python, with backend for Theano library [78], also written in Python. For training the CNNs, we used a NVIDIA GeForce GTX 970, RAM memory of 4GB and 1664 CUDA Cores.

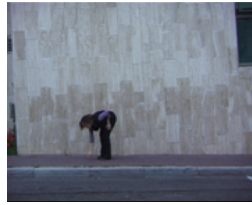
In this Chapter, we will use two public datasets to test the methodology: Weizmann and UCF Sports. We also describe the experiments performed and how the results were validated.

### 5.1 DATASETS

In order to verify the performance of our methodology, we tested our approach in two public action video datasets: Weizmann and UCF Sports.

The Weizmann dataset is provided by the Weizmann Institute of Science's Computer Vision Laboratory [11], which contains 90 image sequences showing nine different people, each one performing 10 natural actions. These actions are Bending (bend), Jumping Jack (jack), Jumping (jump), Jumping in place (pjump), Galloping sideways (side), Running (run), Skipping (skip), Walking (walk), One-hand waving (wave1), Two-hands waving (wave2), as shown in Figure 5.1. All images sequence were recorded in low resolution, with 50 frames per second, at 180 x 144 pixels. In order to evaluate this dataset on our approach we used a Leave-one-out Cross-validation, in which eight actors were used for training and the ninth actor for testing. This was repeated over nine actors and their results were averaged.

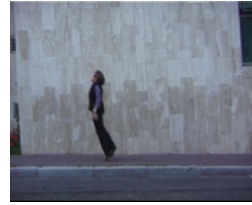
UCF Sports consists of various sports actions collected from broadcast television channels [12]. The dataset includes 10 actions: Diving, Golfing, Kicking, Weightlifting, Horseback-riding, Running, Skateboarding, Swinging 1 (gymnastics, on the pommel horse and floor), Swinging 2 (gymnastics, on the high and uneven bars) and walking, as shown in Figure 5.2. This dataset has a total of 150 sequences with the resolution of 720 x 480 and with 10 frames per second. In order to evaluate this dataset on our approach we used a five-fold cross-validation, in which 1/5 of the data containing all classes is used for testing and the 4/5 are used for training. This was repeated over the 5 configurations and their results were averaged.



(a) Bending



(b) Jumping Jack



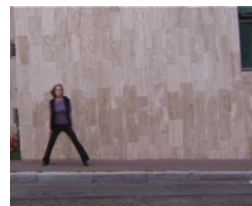
(c) Jumping



(d) Jumping in place



(e) Running



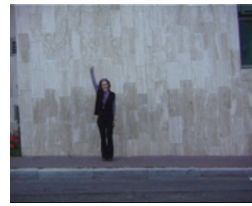
(f) Galloping sideways



(g) Skipping



(h) Walking



(i) One-hand waving



(j) Two-hands waving

Figure 5.1: Natural actions performed in the set of image sequences provided by the Weizmann Institute of Science's Computer Vision Laboratory [11]

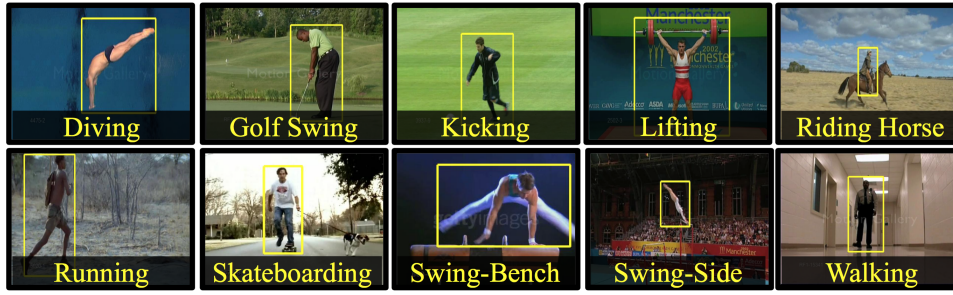


Figure 5.2: Actions performed by video sequences from UCF Sports dataset (illustrated by Soomro and Zamir [12]).

## 5.2 EXPERIMENTS

To produce the final architecture proposed in this work that is based on the outputs obtained by two convolutional neural networks, here called Spatial CNN Stream Spatial and Temporal CNN Stream, we performed some experiments. The two following subsections respectively show the experiments performed.

### 5.2.1 Spatial CNN stream

The experiment 1, whose scheme is illustrated in Figure 5.3, represents our initial proposed Spatial CNN Stream .

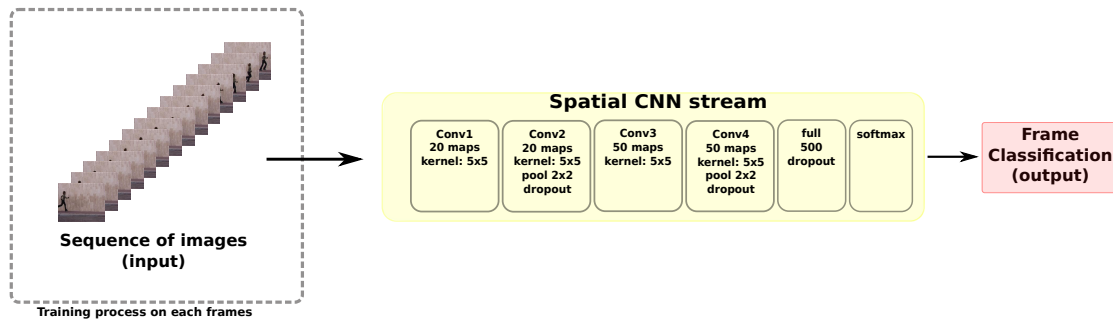


Figure 5.3: Spatial CNN Architecture (Experiment 1).

As we can see in Figure 5.3, our network starts with two layers of Convolution, both with 20 feature maps and a 5x5 kernel, in which the second presents a Pooling layer. This layer was trained using Dropout with probability  $p = 0.25$ . The following two Convolution layers have the same structure, except that they have 50 feature maps each one and then a fully connected layer with 500 neurons (all layers up to the fully connected one used Rectified Linear Unit (ReLU)). Finally, the 10 neurons (representing the 10 classes) with their softmax activation functions and the optimizer used by the network were the Adadelta [63].

We used the validation method to leave a group out, the data belonging to the training group are different from the data of the validation group. The result of this method depends on how

the training and validation data were separated. In this experiment, approximately 70% of the image sequences were used in the training group and 30% in the validation group, that is 60 image sequences for training and 30 for validating our model during training process.

In general, to train convolutional neural networks from scratch a dataset of sufficient size is necessary, because with so many parameters and few training samples, it can easily generate overfitting. We decided to use a CNN pre-trained on a very large dataset (VGG16 [39] network), such as ImageNet, then we removed the last fully-connected layer, that is, the layer with outputs which are the class scores. We fine-tuned the weights of the pre-trained network by back-propagation. We did not train some of the layers, in order to avoid overfitting, and only trained some layers of the network, because lower layers contain more generic features and higher ones, more specific features to the details of the classes from the original dataset used to pre-train the network [75].

As indicated in Figure 4.2, we did not train the first three convolutional blocks (each convolutional block is separated from the others by a MaxPooling layer, see Table 3.2) and trained the last two ones. Then, we trained a multilayer perceptron (MLP) by fine-tuning the network with the new dataset features. Because the Weizmann dataset is quite different from the dataset in which the VGG16 network was pre-trained, we wanted to only use features learned by VGG16 in the lower layers (more generic features, such as edges, curves, etc.) and train the deeper ones because they are more specific to the objects in the pre-trained dataset. With the UCF Sports base we could take some deeper layers, since some of its actions are associated with certain objects (for which the VGG16 network was pre-trained), but we used exactly the same training setup for both datasets, In order to evaluate it in a generic way.

We took a validation dataset from the training dataset to monitor the training and stop it when necessary. Through this validation data, to follow the evolution of the loss function of our optimizer, we used the early-stopping technique to interrupt training process. We followed the loss function with a minimum variation of 0.01 in its value, so that if the error rate did not improve in 20 *epochs*, we would stop training.

The weights of the net were learned using the Stochastic Gradient Descent (SGD) with momentum equal to 0.9 and learning rate is set to  $10^{-4}$ . Fine-tuning was done at a very slow learning rate, with the SGD optimizer instead of an adaptive learning rate optimizer. This was done to ensure that the magnitude of the updates remained very small, so that it did not destroy previously learned features. If the frames of the analyzed dataset had resolution less than 224 x 224, we would use their original dimensions, otherwise we would set them to 224 x 224.

## 5.2.2 Temporal CNN stream

For the temporal network, the Dense Optical Flow was calculated with a pyramid of 3 levels and a scale of 0.5, that is, each next layer of the pyramid is twice smaller than the previous one [6]. For the DOF algorithm we used a window size of 15, polynomial expansion with pixel neighborhood 5 and 3 iterations the algorithm does at each pyramid level.

The temporal network was trained from scratch and its structure is illustrated in Figure 4.4. We used the Adadelata optimizer [63] to train it, which does not need a manual setting of a learning rate. After some tests, we realized that a good approach to generate the stacks of optical flow must be with overlap between stacks equal to 3 ( $o = 3$ ) frames and stack size 10 ( $d = 10$ ), so the format of the input sample of the network is a tensor  $10 \times 224 \times 224$ .

To avoid overfitting, monitored by a set of validation data taken from the training data, we used L2 regularization with parameter  $\lambda = 10^{-4}$  (see Subsection 3.3.1.4), and we also used data augmentation techniques so the network could generalize better. In the temporal one, overfitting began to happen before, probably because the amount of training samples generated from the training dataset was small. So we used L2 regularization with  $\lambda = 10^{-2}$ , and inserted dropout layers with probability 0.25 between the convolutional blocks and 0.5 and 0.6 after the first two fully-connected layers as indicated in Figure 4.4.

### 5.3 EXPERIMENTAL RESULTS

In this section we will analyze the results of experiments with the trained spatial network from scratch, fine-tuned spatial network and temporal network.

#### 5.3.1 Preliminary spatial network

The first experiment was performed with this convolutional network trained with all the frames of each sequence of images belonging to the training data set, where each frame was labeled with the respective class of the sequence of images to which it belonged. The graphic of the loss function during the training already shows overfitting in the initial *epochs* (as observed in the Figure 5.4(b)), because the function starts to increase instead of continuing to decrease its values, which made us stop the training at third *epoch* (see Figure 5.4) and that is when the validation reached an accuracy of **62.87%**.

After the network training with the frames of image sequences, the frames of the test dataset (in this first experiment, the validation dataset was used as test dataset as well, only to check the model for classifying the sequences instead of frames) were tested orderly for each sequence, and the network outputs were observed. The outputs varied through time as the sequence of images was presented. From that data, a histogram was built, and the final decision was to assign the sequence to the most frequent label assigned by the network to the frames, as seen in the histograms of Figure 5.5.

Note that in Figure 5.5(a) jump is the expected class and most frames in the sequence were classified as jump. In the histogram of Figure 5.5(c), all frames were correctly classified (class wave2). In Figure 5.5(b), we note that the sequence was classified incorrectly, instead of pjump, most frames were classified as jack.



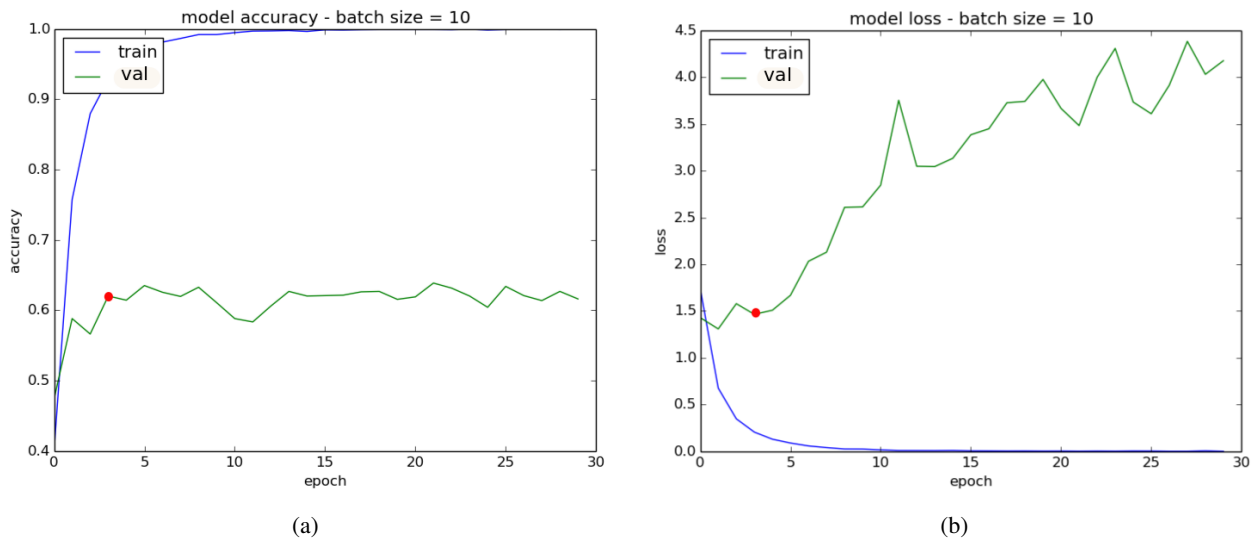


Figure 5.4: (a) Accuracy graphic during training. (b) Loss graphic during training.

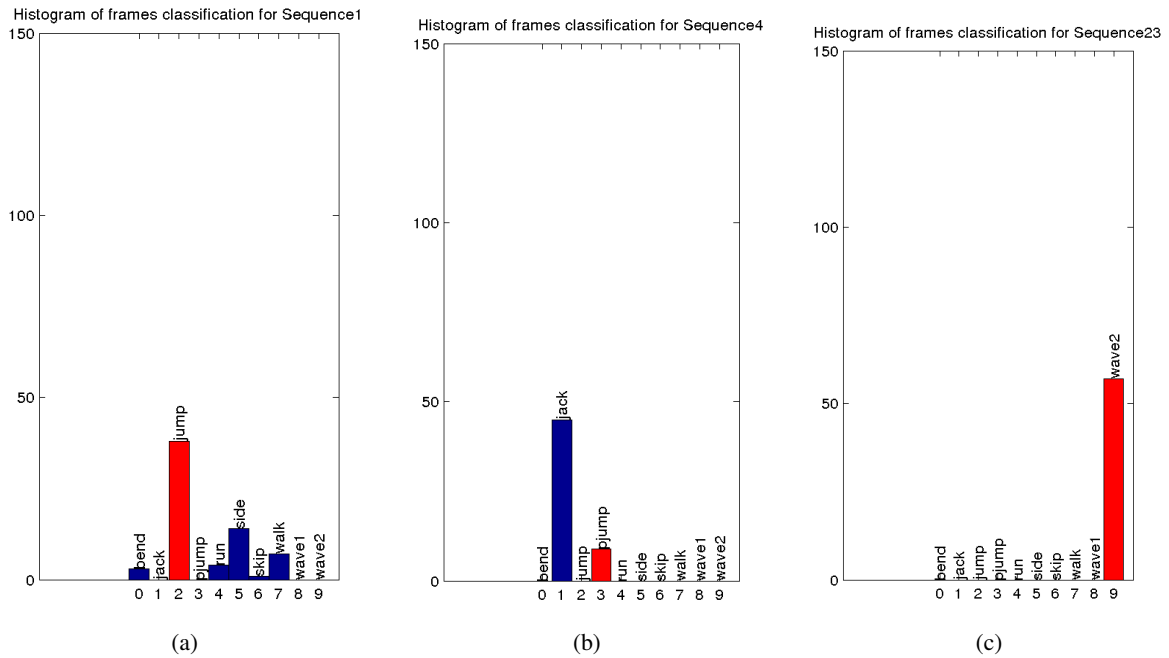


Figure 5.5: (a) Histogram of frames classification for Sequence 1. (b) Histogram of frames classification for Sequence 4. (c) Histogram of frames classification for Sequence 23. The red bar indicates the expected class and the blue bar incorrect classifications

Based on the frequency of correct frames per sequence of images, the network correctly classified 20 out of 30 image sequences, which corresponds to **66.66%** of them. This is a surprising result coming from a action classifier in videos from still images. Based on this, we tried to improve the frame classifier in order to consequently improve the classification of the sequence of images. For this, we kept the same architecture, with the same configurations, but we used the technique of Data Augmentation (in order to apply different types of samples to the network and avoid early overfitting), in which we randomly generated versions of training samples with

horizontal flip and vertical and horizontal shift of the images in 20% of their dimensions. As a result, the accuracy rate has increased.

This improved performance comes from combining a deep architecture, high-capacity model, and an augmented training set [79]. This is reflected in the accuracy of the classification of the model which has increased to **76.67%**.

### 5.3.2 Pre-trained spatial network

When we trained the spatial network in the Weizmann dataset we obtained the individual results presented in Table 5.1. The spatial network reached 84.44% (See Table 5.1), an improved accuracy compared to the first experiment, considering that the classification was made based only on the frequency of correctly classified frames of the sequence of images, which is quite interesting. This rate was due to the fact that some actions may be associated with specific object types and the VGG16 network was pre-trained for a dataset of 1000 different object categories. Also, we fine-tuned the network and trained only high level features and we used the lower layers of the network as descriptors, which were already pre-trained to find edges, curves and other low level features also useful for the Weizmann dataset.

Table 5.1: Accuracies on both Weizmann and UCF datasets on each stream

<b>Dataset</b>	<b>Spatial stream</b>	<b>Temporal stream</b>
Weizmann	84.44%	78.89%
UCF Sports	78.46%	15.38%

When we trained the spatial network in the UCF Sports dataset we obtained the individual results presented also in Table I. The reason we obtained that rate was due to the same reason for Weizmann dataset (We took advantage of the pre-trained VGG16 network for object recognition). We did not pre-process any image, we used raw images as input and the spatial network worked well on frames.

### 5.3.3 Temporal network

For the temporal network we also reached 78.89% (see Table 5.1), but we had a weaker result for the spatial one. This happened because the number of stacks generated by the training dataset was small, so the network failed to generalize very well.

On the other hand, we did an additional test of the UCF Sports dataset on the same temporal stream used for the Weizmann dataset and we obtained a weaker result (15.38%).

This occurred with temporal stream due to the fact that the dataset UCF Sports has sequences with 10 frames per second and the number of stacks generated was less than the number generated by the Weizmann base (50 frames per second). Besides, the different frame rates from the two

datasets made the velocities of motion performed in the videos different and the setup of the algorithm parameters of stacks of Dense Optical Flow would have to be different.

### 5.3.4 Combination of classifiers

When classified by the two-stream method, the rate has greatly improved to 91.11% (see Table 5.2) on Weizmann dataset, which is comparable to published results. Temporal network on UCF SPorts dataset reached 15.38%, which was prejudicial using this stream in combination with the spatial stream. 70.77% of accuracy rate was obtained with the combination of spatial and temporal streams.

Table 5.2: Comparison of accuracies on the Weizmann dataset

Method	Accuracy
<b>Our method (Spatial CNN)</b>	<b>84.44%</b>
Hoai <i>et al.</i> [80]	87.7%
Huang and Wu [81]	88.8%
<b>Our method (combination: Temporal and Spatial CNNs)</b>	<b>91.11%</b>
Zhang <i>et al.</i> [82]	92.89%
Bregonzio <i>et al.</i> [83]	96.66%
Sun <i>et al.</i> [84]	97.8%
Weinland and Boyer [85]	100%

We can observe in Figure 5.6 the confusion matrix for the experiment performed in the Weizmann dataset, confusing only the run class with the skip and walk classes and the skip class with the side class. Since this classification methodology is partly based on frames, we can understand the confusion made between the run, skip and walk classes. This occurs because some frames presented during these movements can be quite similar and thus the classifier can make errors about these classes.

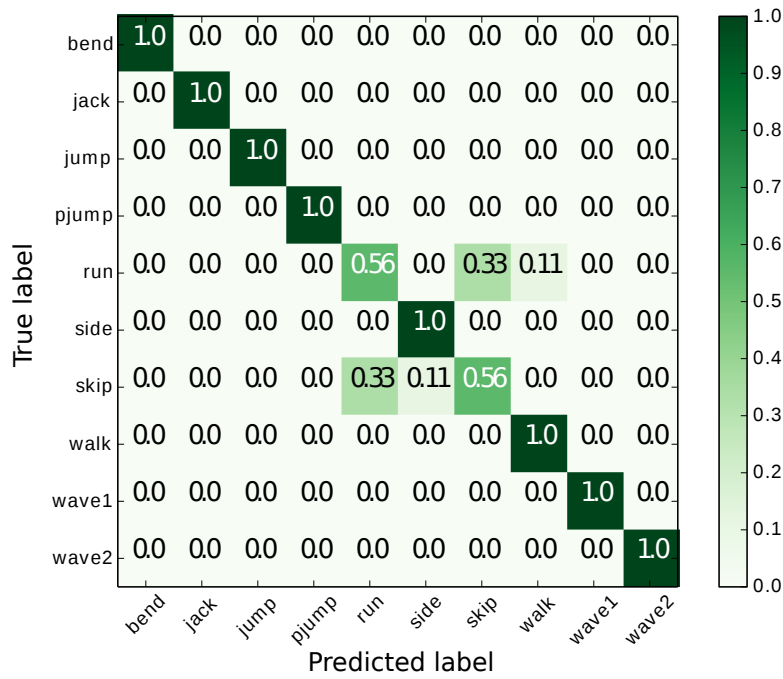


Figure 5.6: Confusion matrix of Weizmann dataset using leave-one-out cross-validation.

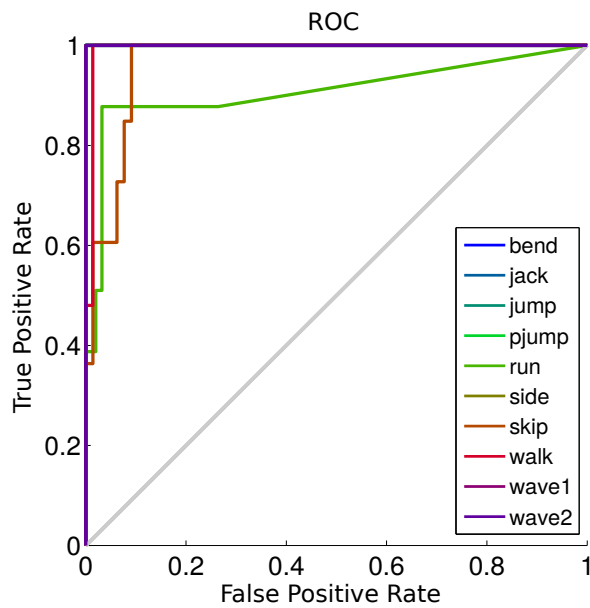


Figure 5.7: ROC curve of Weizmann dataset.

The Receiver Operating Characteristic (ROC) curve for this experiment (see Figure 5.7) also shows the efficiency of the method. **This curve was generated by varying the value of a decision threshold on the outputs of the classifier, evaluating for each case one class against all the others.** In the same way as the confusion matrix, we observed that the classes not correctly classified were run and skip.

Our method achieves an accuracy comparable with published accuracies (91.11%), outperforming [80, 81] (see Table 5.2). And [82, 83, 84, 85] methods present better accuracy rates,

however they use handcrafted descriptors.

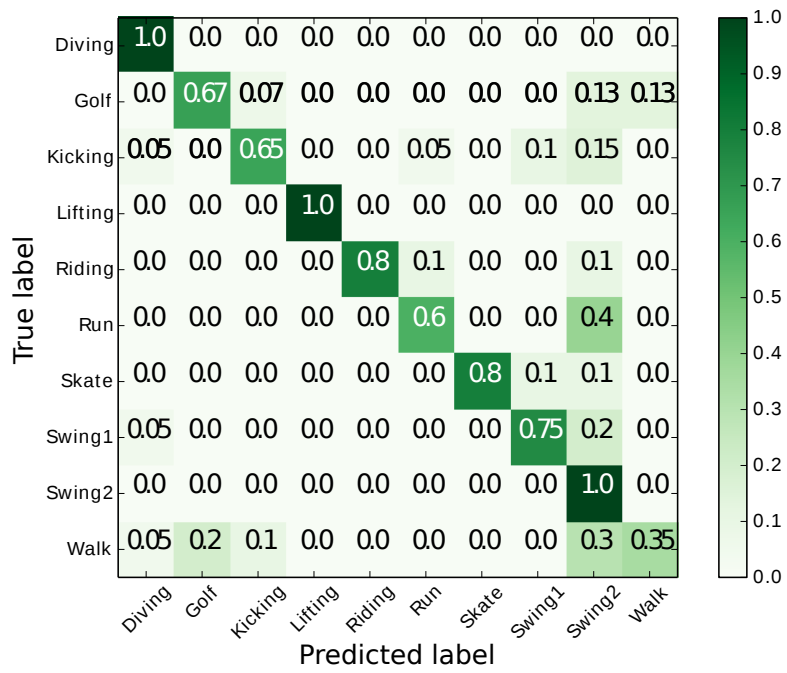


Figure 5.8: Confusion matrix of UCF Sports dataset using leave-one-out cross-validation.

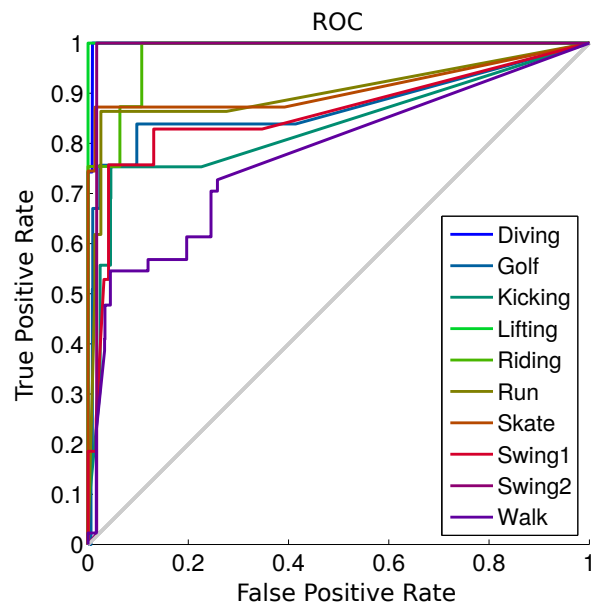


Figure 5.9: ROC curve of UCF Sports dataset.

Table 5.3: Comparison of accuracies on the UCF dataset

Method	Accuracy
Rodriguez <i>et al.</i> [86]	69.2%
<b>Our method (combination: Temporal and Spatial CNNs)</b>	<b>70.77%</b>
<b>Our method (Spatial CNN)</b>	<b>78.46%</b>
Yeffet and Wolf [87]	79.2%
Wang <i>et al.</i> [88]	85.6%

We can observe in Figure 5.8 the confusion matrix for the experiment performed in the dataset UCF Sports, presenting a weaker result than for the dataset Weizmann, occurring confusion between the Golf class and the classes Kicking, Swing2 and Walk and between the Walk and Diving classes, for example. The ROC curve for this experiment (see Figure 5.9) also reflects the classes classified incorrectly. The classes that are farthest from the top left of the chart are the Golf, Kicking, Run and Walk classes.

We outperform only Rodriguez *et al.* [86] (see Table 5.3), but Yeffet and Wolf [87] use local trinary patterns and Wang *et al.* [88] use 3D-HOG descriptors, whereas we do not perform any pre-processing, using raw images as input to our spatial stream.

In the next chapter we conclude this dissertation with the final considerations, the main contributions of this research work and the suggestions for future work.

## 6 CONCLUSIONS AND FUTURE WORKS

In this work, an architecture based on a two-stream CNN was investigated, in which we have each frame of the sequence of images (Spatial features) as an input training sample of one CNN, another CNN in which the inputs are the stacked dense optical flow components between the frames of the sequence (Temporal features).

The Spatial network stream uses a 2D convolutional architecture named VGG16, pre-trained on the largest and most challenging ILSVRC-2014 dataset [39], which have achieved good accuracy rates in object recognition.

These two structures were trained separately and we generated histograms to make the sequence classification on each network based on the most frequent class from the sequence. For the final classification, both histograms were normalized and were summed to produce a single output.

Initially, we trained a convolutional network from zero with the frames of its image sequences, in order to analyze if the network could classify only from frames of the sequence. We observed that the outputs varied through time as the sequence of images was presented. From that data, a histogram was built, and the final decision was to assign the sequence to the most frequent label assigned by the network to the frames. With this, we reached 66.66% of accuracy on Weizmann dataset.

Based on the hypothesis that the network was able to classify the videos from their frames, we worked on the classifier improvement. Applying Data Augmentation technique, in which we randomly generated versions of training samples with horizontal flip and vertical and horizontal shift of the images, we increased the classification accuracy to 76.67% on Weizmann dataset.

We realized that in general to train a convolutional neural network requires a reasonable amount of data. Whereas the Weizmann and UCF Sports datasets are not as large, we fine-tuned the weights of the pre-trained (pre-trained VGG16 network [39]) network by back-propagation. Thus, we have achieved an accuracy rate of 84.44% on Weizmann dataset. We also trained from scratch a temporal convolutional network with stacks of Dense Optical Flow (DOF), and on Weizmann dataset we reached an accuracy of 78.89%. Combining spatial and temporal classifiers we obtained 91.11% on Weizmann dataset.

We showed that still frames belonging to a certain sequence of images curiously make it possible to classify the action performed in such a sequence. We believe that, since the VGG16 network was pre-trained for a dataset of 1000 classes of different objects and some actions are associated with certain types of objects, this contributed significantly to the learning of the spatial network. This indicates that the transfer learning technique was used efficiently to recognize human actions, using a previously trained network to recognize objects.

Even though the results with the temporal stream in the dataset UCF Sports were not the best, the result with the spatial stream is comparable to published results and the temporal stream can have improvements in its architecture through the number of epochs, training optimization method and network depth.

As future work, we intend to further explore the capacity of learning actions from single frames, combining our spatial network with other classifiers. Also, make our architecture more generic for videos with different frames per second rates and try it with even more challenging datasets such as UCF-101 [89] and HMDB-51 [90].



## REFERENCES

- [1] R. C. Gonzalez and R. E. Woods, *Digital image processing*, 3rd ed. Prentice Hall, Boston, MA, USA, 2007.
- [2] R. Minetto, “Detecção robusta de movimento de camera em videos por analise de fluxo otico ponderado,” Ph.D. dissertation, Universidade Estadual de Campinas, 2007.
- [3] K. Simonyan and A. Zisserman, “Two-stream convolutional networks for action recognition in videos,” in *Advances in Neural Information Processing Systems*, 2014, pp. 568–576.
- [4] S. S. Beauchemin and J. L. Barron, “The computation of optical flow,” *ACM computing surveys (CSUR)*, vol. 27, no. 3, pp. 433–466, 1995.
- [5] B. K. Horn and B. G. Schunck, “Determining optical flow,” *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.
- [6] G. Farneback, “Two-frame motion estimation based on polynomial expansion,” in *Scandinavian conference on Image analysis*. Springer, 2003, pp. 363–370.
- [7] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.
- [8] S. S. Haykin, *Neural networks and learning machines*. Pearson Upper Saddle River, NJ, USA:, 2009, vol. 3.
- [9] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, “Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations,” in *Proceedings of the 26th annual international conference on machine learning*. ACM, 2009, pp. 609–616.
- [10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [11] M. Blank, L. Gorelick, E. Shechtman, M. Irani, and R. Basri, “Actions as space-time shapes,” in *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*, vol. 2. IEEE, 2005, pp. 1395–1402.
- [12] K. Soomro and A. R. Zamir, “Action recognition in realistic sports videos,” in *Computer Vision in Sports*. Springer, 2014, pp. 181–208.
- [13] L.-P. Morency, A. Quattoni, and T. Darrell, “Latent-dynamic discriminative models for continuous gesture recognition,” in *2007 IEEE conference on computer vision and pattern recognition*. IEEE, 2007, pp. 1–8.
- [14] A. Dehghan, O. Oreifej, and M. Shah, “Complex event recognition using constrained low-rank representation,” *Image and Vision Computing*, vol. 42, pp. 13–21, 2015.

- [15] A. Yilmaz, O. Javed, and M. Shah, “Object tracking: A survey,” *Acm computing surveys (CSUR)*, vol. 38, no. 4, p. 13, 2006.
- [16] L. Wang, Y. Qiao, and X. Tang, “Action recognition with trajectory-pooled deep-convolutional descriptors,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4305–4314.
- [17] X. Peng, L. Wang, X. Wang, and Y. Qiao, “Bag of visual words and fusion methods for action recognition: Comprehensive study and good practice,” *Computer Vision and Image Understanding*, 2016.
- [18] M. Hasan and F. Boris, “Svm: Machines à vecteurs de support ou séparateurs à vastes marges,” *Rapport technique, Versailles St Quentin, France. Cité*, p. 64, 2006.
- [19] H. Wang and C. Schmid, “Action recognition with improved trajectories,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 3551–3558.
- [20] H. Baya, A. Essa, T. Tuytelaarsb, and L. Van Gool, “Speeded-up robust features (surf),” *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.
- [21] S. Sadanand and J. J. Corso, “Action bank: A high-level representation of activity in video,” in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 1234–1241.
- [22] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, “Large-scale video classification with convolutional neural networks,” in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2014, pp. 1725–1732.
- [23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [24] T. Brox, A. Bruhn, N. Papenber, and J. Weickert, “High accuracy optical flow estimation based on a theory for warping,” in *European conference on computer vision*. Springer, 2004, pp. 25–36.
- [25] D. Fleet and Y. Weiss, “Optical flow estimation,” in *Handbook of mathematical models in computer vision*. Springer, 2006, pp. 237–257.
- [26] C. Feichtenhofer, A. Pinz, and A. Zisserman, “Convolutional two-stream network fusion for video action recognition,” *arXiv preprint arXiv:1604.06573*, 2016.
- [27] A. Klaser, M. Marszałek, and C. Schmid, “A spatio-temporal descriptor based on 3d-gradients,” in *BMVC 2008-19th British Machine Vision Conference*. British Machine Vision Association, 2008, pp. 275–1.

- [28] P. Scovanner, S. Ali, and M. Shah, “A 3-dimensional sift descriptor and its application to action recognition,” in *Proceedings of the 15th ACM international conference on Multimedia*. ACM, 2007, pp. 357–360.
- [29] G. Willems, T. Tuytelaars, and L. Van Gool, “An efficient dense and scale-invariant spatio-temporal interest point detector,” in *European conference on computer vision*. Springer, 2008, pp. 650–663.
- [30] E. E. Aksoy, A. Abramov, J. Dörr, K. Ning, B. Dellen, and F. Wörgötter, “Learning the semantics of object–action relations by observation,” *The International Journal of Robotics Research*, vol. 30, no. 10, pp. 1229–1249, September 2011.
- [31] Q. V. Le, W. Y. Zou, S. Y. Yeung, and A. Y. Ng, “Learning hierarchical invariant spatio-temporal features for action recognition with independent subspace analysis,” in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 3361–3368.
- [32] M. Baccouche, F. Mamalet, C. Wolf, C. Garcia, and A. Baskurt, “Spatio-temporal convolutional sparse auto-encoder for sequence classification.” in *BMVC*, 2012, pp. 1–12.
- [33] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler, “Convolutional learning of spatio-temporal features,” in *European conference on computer vision*. Springer, 2010, pp. 140–153.
- [34] S. Ji, W. Xu, M. Yang, and K. Yu, “3d convolutional neural networks for human action recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2013.
- [35] H. Wang, H. Zhou, and A. Finn, “Discriminative dictionary learning via shared latent structure for object recognition and activity recognition,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 6299–6304.
- [36] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, vol. 1. IEEE, 2005, pp. 886–893.
- [37] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, “Learning spatiotemporal features with 3d convolutional networks,” in *2015 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2015, pp. 4489–4497.
- [38] S. Zha, F. Luisier, W. Andrews, N. Srivastava, and R. Salakhutdinov, “Exploiting image-trained cnn architectures for unconstrained video classification,” *arXiv preprint arXiv:1503.04144*, 2015.
- [39] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

- [40] T. Morris, *Computer vision and image processing*. Palgrave Macmillan, 2004.
- [41] M. Sonka, V. Hlavac, and R. Boyle, *Image processing, analysis, and machine vision*. Cengage Learning, 2014.
- [42] D. A. Forsyth and J. Ponce, “A modern approach,” *Computer Vision: A Modern Approach*, pp. 88–101, 2003.
- [43] B. McCane, K. Novins, D. Crannitch, and B. Galvin, “On benchmarking optical flow,” *Computer Vision and Image Understanding*, vol. 84, no. 1, pp. 126–143, 2001.
- [44] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, “Performance of optical flow techniques,” *International journal of computer vision*, vol. 12, no. 1, pp. 43–77, 1994.
- [45] B. Galvin, B. McCane, K. Novins, D. Mason, S. Mills *et al.*, “Recovering motion fields: An evaluation of eight optical flow algorithms.” in *BMVC*, vol. 98, 1998, pp. 195–204.
- [46] B. D. Lucas, “Generalized image matching by the method of differences,” 1985.
- [47] B. D. Lucas, T. Kanade *et al.*, “An iterative image registration technique with an application to stereo vision,” 1981.
- [48] J.-Y. Bouguet, “Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm,” *Intel Corporation*, vol. 5, no. 1-10, p. 4, 2001.
- [49] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library*. " O’Reilly Media, Inc.", 2008.
- [50] C. Tomasi and T. Kanade, “Detection and tracking of point features,” 1991.
- [51] G. Farneback, “Polynomial expansion for orientation and motion estimation,” Ph.D. dissertation, Linköping University Electronic Press, 2002.
- [52] T. M. Mitchell, “Machine learning,” *Burr Ridge, IL: McGraw Hill*, vol. 45, 1997.
- [53] S. Harnad, “The annotation game: On turing (1950) on computing, machinery, and intelligence,” *The Turing test sourcebook: philosophical and methodological issues in the quest for the thinking computer*, 2006.
- [54] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*. Prentice hall Upper Saddle River, 2003, vol. 2.
- [55] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.
- [56] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.” *Psychological review*, vol. 65, no. 6, p. 386, 1958.
- [57] Y. Chauvin and D. E. Rumelhart, *Backpropagation: theory, architectures, and applications*. Psychology Press, 1995.

- [58] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48.
- [59] Y. LeCun *et al.*, “Generalization and network design strategies,” *Connectionism in perspective*, pp. 143–155, 1989.
- [60] K. Jarrett, K. Kavukcuoglu, Y. LeCun *et al.*, “What is the best multi-stage architecture for object recognition?” in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 2146–2153.
- [61] P. Golik, P. Doetsch, and H. Ney, “Cross-entropy vs. squared error training: a theoretical and experimental comparison.” in *Interspeech*, 2013, pp. 1756–1760.
- [62] H. Robbins and S. Monro, “A stochastic approximation method,” *The annals of mathematical statistics*, pp. 400–407, 1951.
- [63] M. D. Zeiler, “Adadelta: an adaptive learning rate method,” *arXiv preprint arXiv:1212.5701*, 2012.
- [64] D. Yu, L. Deng, and D. Yu, “Deep learning methods and applications,” *Foundations and Trends in Signal Processing*, 2014.
- [65] Y. Bengio, I. J. Goodfellow, and A. Courville, “Deep learning,” *An MIT Press book in preparation. Draft chapters available at <http://www.iro.umontreal.ca/bengioy/dlbook>*, 2015.
- [66] P. O. Glauner, “Deep convolutional neural networks for smile recognition,” *arXiv preprint arXiv:1508.06535*, 2015.
- [67] H. A. Song and S.-Y. Lee, “Hierarchical representation using nmf,” in *International Conference on Neural Information Processing*. Springer, 2013, pp. 466–473.
- [68] B. A. Olshausen *et al.*, “Emergence of simple-cell receptive field properties by learning a sparse code for natural images,” *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.
- [69] L. Gomes, “Machine-learning maestro michael jordan on the delusions of big data and other huge engineering efforts,” *IEEE Spectrum*, Oct, vol. 20, 2014.
- [70] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, “Imagenet large scale visual recognition challenge,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.
- [71] Y. Bengio, “Deep learning of representations: Looking forward,” in *International Conference on Statistical Language and Speech Processing*. Springer, 2013, pp. 1–37.
- [72] D. Scherer, A. Müller, and S. Behnke, “Evaluation of pooling operations in convolutional architectures for object recognition,” in *International Conference on Artificial Neural Networks*. Springer, 2010, pp. 92–101.

- [73] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting.” *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.
- [74] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.
- [75] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” in *Advances in neural information processing systems*, 2014, pp. 3320–3328.
- [76] W. Garage, “Opencv,” *Open Source Computer Vision Library*.(Accessed 2010) Available at: <http://opencv.willowgarage.com>, 2014.
- [77] F. Chollet, “Keras,” <https://github.com/fchollet/keras>, 2015.
- [78] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: <http://arxiv.org/abs/1605.02688>
- [79] J. Salamon and J. P. Bello, “Deep convolutional neural networks and data augmentation for environmental sound classification,” *IEEE Signal Processing Letters*, vol. 24, no. 3, pp. 279–283, 2017.
- [80] M. Hoai, Z.-Z. Lan, and F. De la Torre, “Joint segmentation and classification of human actions in video,” in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 3265–3272.
- [81] W. Huang and Q. J. Wu, “Human action recognition based on self organizing map,” in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2130–2133.
- [82] Z. Zhang, Y. Hu, S. Chan, and L.-T. Chia, “Motion context: A new representation for human action recognition,” *Computer Vision–ECCV 2008*, pp. 817 – 829, 2008.
- [83] M. Bregonzio, S. Gong, and T. Xiang, “Recognising action as clouds of space-time interest points,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1948 – 1955.
- [84] X. Sun, M. Chen, and A. Hauptmann, “Action recognition via local descriptors and holistic features,” in *Computer Vision and Pattern Recognition Workshops, 2009. CVPR Workshops 2009. IEEE Computer Society Conference on*. IEEE, 2009, pp. 58 – 65.
- [85] D. Weinland and E. Boyer, “Action recognition using exemplar-based embedding,” in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1 – 7.

- [86] M. D. Rodriguez, J. Ahmed, and M. Shah, “Action mach a spatio-temporal maximum average correlation height filter for action recognition,” in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1– 8.
- [87] L. Yeffet and L. Wolf, “Local trinary patterns for human action recognition,” in *Computer Vision, 2009 IEEE 12th International Conference on*. IEEE, 2009, pp. 492 – 497.
- [88] H. Wang, M. M. Ullah, A. Klaser, I. Laptev, and C. Schmid, “Evaluation of local spatio-temporal features for action recognition,” in *BMVC 2009-British Machine Vision Conference*. BMVA Press, 2009, pp. 124 – 1.
- [89] K. Soomro, A. R. Zamir, and M. Shah, “Ucf101: A dataset of 101 human actions classes from videos in the wild,” *arXiv preprint arXiv:1212.0402*, 2012.
- [90] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre, “Hmdb: a large video database for human motion recognition,” in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2556–2563.