UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Hugo Aparecido de Lima França Chaves

Filter Learning from Deep Descriptors of Fully Convolutional Siamese Network for Tracking in Videos

Juiz de Fora

2019

UNIVERSIDADE FEDERAL DE JUIZ DE FORA
INSTITUTO DE CIÊNCIAS EXATAS
PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Hugo Aparecido de Lima França Chaves

Filter Learning from Deep Descriptors of Fully Convolutional Siamese Network for Tracking in Videos

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Orientador: Marcelo Bernardes Vieira
Coorientador: Augusto Santiago Cerqueira

Juiz de Fora
2019

Hugo Aparecido de Lima França Chaves

Filter Learning from Deep Descriptors of Fully Convolutional Siamese Network for Tracking in Videos

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação, do Instituto de Ciências Exatas da Universidade Federal de Juiz de Fora como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Aprovada em 3 de Junho de 2019.

BANCA EXAMINADORA

Prof. D.Sc. Marcelo Bernardes Vieira - Orientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Augusto Santiago Cerqueira - Coorientador
Universidade Federal de Juiz de Fora

Prof. D.Sc. Saulo Moraes Villela
Universiade Federal de Juiz de Fora

Prof. Ph.D. Hélio Pedrini
Universidade Estadual de Campinas

Dedico este trabalho aos meus pais, Euzimar e Mary Jamel e à minha irmã, Juliana.

ACKNOWLEDGMENTS

Eu gostaria de agradecer meus pais, Euzimar e Mary Jamel, pelo esforço e empenho que depositaram a favor da minha formação; À Juliana pelo carinho que tem comigo; À Priscila pelo carinho e apoio ao longo do mestrado; Ao professor Marcelo por ter orientado este trabalho e minha formação ao longo do mestrado e ter me acolhido no PGCC; Ao professor Augusto por também ter orientado este trabalho e ter contribuído com minha formação desde a graduação; Ao Kevyn pelo auxílio na execução deste trabalho; Ao André e ao Hermerson pela parceria ao longo do mestrado; Aos demais professores do PGCC que contribuíram com minha formação e a todas pessoas que contribuiram de alguma forma também para a minha conclusão.

"A dúvida não é uma condição agradável, mas a certeza é absurda." Voltaire

RESUMO

Nos últimos anos, os avanços em Aprendizado Profundo revolucionaram diversas subáreas da Visão Computacional, incluindo o Rastreamento de Objetos Visuais. Um tipo especial de rede neural profunda, a Rede Neural Siamesa, chamou a atenção da comunidade especializada em rastreamento. Ela possui baixo custo computacional e alta eficácia para comparar a similaridade entre objetos. Atualmente, a comunidade científica atingiu resultados notáveis ao aplicar tais redes ao problema de Rastreamento de Objetos Visuais. No entanto, observou-se que limitações dessa rede neural impactam negativamente no rastreamento. Superou-se o problema ao se obter um novo descritor para referência do objeto combinando descritores passados fornecidos pelo rastreador. Em particular, foi proposto a combinação de sinal de descritores em blocos de memórias de longo e de curto prazo, os quais representam a primeira e a mais recente aparência do objeto, respectivamente. Um descritor final é gerado a partir desses blocos de memória, o qual o rastreador usa como referência. Este trabalho enfatizou-se na obtenção de um método para calcular um banco de filtros otimizado através do uso de um algoritmo genético. O banco de filtros é utilizado então para gerar a saída da memória de curto prazo. De acordo com experimentos realizados na base de dados OTB, esta proposta apresenta ganhos em comparação com a proposta original da *SiamFC*. Considerando a métrica *área abaixo da curva*, há ganhos de 7.4% e 3.0% para os gráficos de precisão e sucesso, respectivamente, tornando este trabalho comparável a métodos do estato da arte.

Palavras-chave: Rastreamento. Redes Siamesas. Descritores Profundos.

ABSTRACT

In recent years, the advancement of Deep Learning has revolutionized many areas in Computer Vision, including Visual Object Tracking. A particular type of deep neural network, the Siamese Neural Network, brought the attention of Visual Object Tracking community. This neural network has a relatively low computational cost, and high efficacy framework used to compare the similarity between objects. Nowadays, the scientific community achieved remarkable success applying such frameworks in the tracking problem. However, the limitations this neural network impact negatively in its performance. We overcome this problem by obtaining a new descriptor for the reference object combining past descriptors outputted from the tracker. Specifically, we propose a combination of the signal of descriptors in *long* and *short term memory* blocks, which represent the first and the recent appearance of the object, respectively. A final descriptor is composed of such memory blocks, and the tracker uses it as a reference. In particular, this work emphasized in the obtention of a method to compute an optimized filter bank through the usage of a genetic algorithm. The filter bank is then used to compute the short term memory output. According to experiments performed in the widely used OTB dataset, our proposal improves the baseline performance. The improvements for the *area under the curve* metrics are 7.4% and 3.0%, for precision and success plots, respectively, being comparable to the state-of-the-art methods.

Keywords: Tracking. Siamese Network. Deep Descriptors.

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

h	Generic system or filter.
$x[n]$	Input signal in time n.
$y[n]$	Output signal in time n.
q	Multiplicative constant of a system.
L_D	Delay operator.
$g[n]$	Generic signal in time n.
w_x	Weight of the x^{th} element of a neural network.
a_x	x^{th} input of a neuron.
z	Output of a neural network.
b	Bias of a neural network.
$\Phi(.)$	Activation Function .
$max(.)$	Maximum between two values.
$a_l^{k_l}$	Input Feature map at layer l composed by k_l filters.
$w_{l,kl}^{k_{l'}}$	$k_{l'}^{th}$ filter at layer l of the k_l^{th} channel.
$z_{l'}^{k_{l'}}$	Output Feature map at layer l' composed by $k_{l'}$ filters.
$b_l^{k_{l'}}$	Bias for the $k_{l'_{th}}$ filter in layer l.
s_l	Stride in layer l.
γ	Atomic size of a Fully-Convolutional Neural Network.
L	Translation operator.
I	Input image.
s	Stride.
d	Size of a feature map.
$\boldsymbol{z}$	Encoded vector $\boldsymbol{z} \in \mathbb{R}^K$.
f	Encoding function, $f : \mathbb{R}^2 \to \mathbb{R}^K$.
l	Similarity function $l : \mathbb{R}^K \to \mathbb{R}$.
S	Similarity.

$\mathbf{z}[n]$	Encoded signal of descriptors from bounding box outputted in time n.
BB_n	n^{th} bounding Box in a video outputted by the tracker.
$\mathbf{g}[n]$	Encoded signal of descriptors from ground-truth in time n.
GT_n	n^{th} region of interest in a video indicated by the ground-truth.
$\mathbf{z}_{comb}[n]$	Combined Descriptor signal.
$\mathbb{O}(.)$	Generic operation between feature vectors.
$\mathbf{z}_{long}$	Long Term Memory signal.
$E(.)$	Expected value operator.
Q	First Q frames of a videos.
$\mathbf{h}_{LTI}[n]$	Vector of LTI filters at coeficient n.
$h_{LTI_k}[n]$	k^{th} component of a Vector of LTI filters at coeficient n.
q_k	k^{th} coeficient of a system or filter.
$\mathbf{z}_{short}[n]$	Short Term Memory signal.
$\mathbf{z}_{ext}[n]$	Signal of extended video
$\mathbb{M}(.)$	Generic memory combination.
α	Conservative factor.
p_g	Probability of gene mutation.
p_m	Probability of selecting an individual for mutation.
p_c	Probability of crossover.
P	Population size.
M_i	Maximum number of generations.
F	F-Measure.
p	Precision.
r	Recall.
Γ	F-Measure threshold.
$\boldsymbol{\sigma}[n]$	White noise vector at time n.
t	t^{th} frame of a video.
T	Total number of frames of a video.

LIST OF ACRONYMS

ALIEN Appearance Learning In Evidential Nuisance

AUC Area under the curve

BB Bounding Box

C-COT Continuous Convolution Operator Tracker

CFN Correlation Filter Network

CNN Convolutional Neural Network

CV Computer Vision

DCFNet Discriminant Correlation Filters Network

DL Deep Learning

DLT Deep Learning Tracker

DNN Deep Neural Network

DSiam Dynamic Siamese network

DSP Digital Signal Processing

ECO Efficient Convolution Operators

FCNN Fully Convolutional Neural Network

FFT Fast Fourier Transform

FPS Frames Per Second

GA Genetic Algorithm

GOTURN Generic Object Tracking Using Regression Networks

GT Ground-truth

IoU Intersection-over-Union

KCF Kernelized Correlation Filter

LE Location Error

LSTM Long Short-Term Memory

LTI Linear Time-Invariant

MDNet Multi-Domain Network

ML Machine Learning

NCC Normalized Cross Correlation

NN Neural Network

OS Overlap Score

OTB Online Tracking benchmark

RNN Recurrent Neural Network

RoI Region of Interest

SIFT Scale Invariant Feature Transform

SINT Siamese INstance search Tracker

SNN Siamese Neural Network

TLD Tracking-Learning-Detection

VOT Visual Object Tracking

CONTENTS

1 INTRODUCTION

Visual Object Tracking (VOT) is one of the fundamental problems in Computer Vision (CV). It consists of providing an object's trajectory along with a video given its initial coordinates. VOT is a key component in many applications such as surveillance system, robotics, autonomous driving, intelligent traffic control, augmented reality, sports video analysis (DANELLJAN et al., 2016; WANG; YEUNG, 2013). The traditional approaches for this problem rely on mathematical models designed to detect and extract features of a target. These features compose an appearance model used to discriminate the target against the background (XIANG, 2011). The features are manually designed, or *hand-crafted*, beforehand by human experts for particular issues as occlusions, illumination changes, and many others. The progress accomplished in recent years in Machine Learning (ML) area - specially Deep Learning (DL) - has empowered impressive advancement to CV area. Since the astonishing performance of Krizhevsky et al. (2012) in ILSVRC 2012, CV community has made a notorious effort to obtain feature descriptors based on data instead of those designed by experts (NANNI et al., 2017). Such descriptors are obtained through Deep Neural Network (DNN) and are known as *Deep Descriptors*.

A specific type of Convolutional Neural Network (CNN) employed in feature design task is called Siamese Neural Network (SNN). It is dedicated for template comparison and has been used for a wide range of applications, including signature verification, audio analysis, face recognition, and VOT (BROMLEY et al., 1994; TAIGMAN et al., 2014; MANOCHA et al., 2018). The capability to generate descriptors for similarity comparison between a reference and a candidate image has inspired works based on this topology for VOT (VALMADRE et al., 2017).

This work tackles the VOT problem in a multidisciplinary approach that comprises concepts of Digital Signal Processing (DSP), Genetic Algorithm (GA) and Neural Networks (NNs), where we improve the performance of a specific type of SNN dedicated for tracking proposed by Bertinetto et al. (2016). Our premise is that combining the descriptor outputted by the SNN along the video could result in significant performance improvement for tracking. Moreover, it will be shown that filtering the descriptors provided by the network over time combined with reliable Bounding Box (BB) obtained from

the video is a feasible way to achieve a combination that outperforms the isolated SNN dedicated for tracking.

1.1 MOTIVATION

This proposal was motivated by the prior expertise of our research group in this area (MAIA et al., 2016; MAIA H.A.; OLIVEIRA, 2016) and advancement of DL techniques and its applications to CV, particularly for VOT. The usage of such ML tool has provided VOT systems the capability of extracting semantic information from data (ZAGORUYKO; KOMODAKIS, 2015) that justify why DL systems outperforms classical approaches for VOT and its variations (KALAL et al., 2012; PERNICI; BIMBO, 2014).

Notably, the recent proposal of deep SNNs in the literature has presented improvements in VOT performance. An SNN maps similar objects into similarity feature vectors, which make it easy to compare an object to a given image. However, observing the behavior of a particular SNN, the SiamFC, it is possible to notice a mapping limitation of its output: a minimal variation at the network's input results in a considerable different descriptor at the output. One solution for this problem is presented by Bertinetto et al. (2016), who suggests a simple way to overcome this issue taking information from 36 descriptors close to the target instead of a single descriptor. Additionally, Valmadre et al. (2017) suggested that moving average filtering of the descriptors outputted by the SNN can lead to favorable results in object mapping for VOT purposes. As DL frameworks face some obstacles to its training process, for instance: the scarcity of labeled data and high computational cost, we decided to overcome the SNN limitation complementing the object mapping through a temporal combination of descriptors.

1.2 PROBLEM DEFINITION

The tracking problem can be defined as follows: taking the BB from the first frame, an algorithm should output the corresponding object's BB on all the following frames of a video as depicted in Figure 1.1.

A VOT algorithm has to overcome several challenges, such as illumination changes, partial occlusion, abrupt motion, pose variation, background clutter, and several other peculiarities. For this work, we assume that occlusion does not last in the long term, and

Figure 1.1: Example of VOT in a video.

then a temporal relationship between frames can be maintained, disregarding occlusion detection strategies.

1.3 OBJECTIVES

The primary objective of this work is to show that for a tracking dedicated SNN, the combination of descriptors extracted from the outputted BB in each frame can outperform traditional usage of the SNN. Moreover, we demonstrate that filtering the descriptors contributes to improving the system's performance, where the filters' kernels are learned in a supervised way.

A secondary objective is to show the feasibility of employing a GA in few videos to obtain a filter bank used for tracking.

1.4 CONTRIBUTIONS

The significant contributions of this work result from the hypothesis that guided this research, it is: the combination of deep descriptors extracted from the output of a tracking dedicated SNN can be used to improve the system's performance.

The first contribution is the confirmation of the hypothesis previously stated. We compared the baseline performance to our proposal, which complements the SiamFC using a particular combination of descriptors.

Additionally, another highlight is the combination itself. The proposed method comprises a combination of descriptors obtained through Linear Time-Invariant (LTI) filter bank, computed via GA, (*Short Term Memory*) and reliable ones obtained by the tracker (*Long Term Memory*). Finally, the present proposal has low complexity at runtime when compared to DNNs. Therefore, it requires low marginal computation resources, which would make it useful for real-time applications.

1.5 OUTLINE

Chapter 2 provides the reader with a fundamental understanding of the general requirements of this work. In Chapter 3, we briefly present the classical approaches for VOT and filter optimization, the introduction of DL in the literature, and its application for VOT. In Chapter 4, we present the proposal of this work to complement the SiamFC through a combination of descriptors. Chapter 5 shows the experiments performed to verify the suitability of the proposed method and its impact in tracking performance according to a benchmark. Finally, in Chapter 6, we present the final remarks of this work, its contribution, and suggestions for future work.

2 FUNDAMENTALS

This chapter presents some basic concepts to help the reader to better understand this text. As this is a multidisciplinary work, it is introduced some key components of *Signals and Systems* (Section 2.1), *Genetic Algorithms* (Section 2.2) and *Neural Networks* (Section 2.3).

2.1 SIGNALS AND SYSTEMS

A signal is a function of one or more independent variables which has information about the behavior or nature of a phenomenon (OPPENHEIM et al., 1996). Few examples of a signal are: a sound, which represents the air pressure variation over time; an image, which shows different brightness levels for points in a two-dimensional space or even the price of a given stock over time in a financial market.

A system is what transforms a particular signal into another one or any desired response (OPPENHEIM et al., 1996). An environment that modifies a person's voice, like a cave that generates an echo, is an example of a system. Another example is a camera that transforms the light from an environment into photography; A mathematical operational as a moving average over a stock price can also be understood as a system.

Therefore, the interaction between a signal and a system can be mathematically expressed in Equation (2.1):

$$x[n] \xrightarrow{h} y[n], \tag{2.1}$$

where $x[n]$ represents the input signal over its independent variable n, h represents the system, that operates over the input, and the output $y[n]$ is the result of the interaction between the $x[n]$ and h. We emphasize our analysis to systems that present the property of *linearity* and *time-invariance*.

2.1.1 LINEARITY

According to Oppenheim et al. (1996), for a system whose response to the inputs $x_1[n]$ and $x_2[n]$ are the outputs $y_1[n]$ and $y_2[n]$, respectivaly, the *superposition* property is verified if

it has the additivity and the homogeneity properties, that are described as:

1. Additivity: The response to $x_1[n] + x_2[n]$ is $y_1[n] + y_2[n]$.

2. Homogeneity: The response to $q \cdot x_1[n]$ is $q \cdot y_1[n]$.

   If a system allows the superposition, then it is a *linear system*.

2.1.2 TIME-INVARIANCE

A system is classified as *time-invariant* if it remains unchanged over time. This property can be verified using a delay operator, which is defined as follows:

Let L_D be a delay operator for a given signal $g[n]$. Thus, the delayed signal is represented as shown by:

$$L_D g[n] = g[n - D] \tag{2.2}$$

Then, a system is *time-invariant* if the delay operator L_D is applied over the output $y[n]$ (Eq. 2.3a) of the system h generates the same result than if it was applied to the input signal (Eq. 2.3b), i.e., the system is *time-invariant* if Equation (2.3c) holds.

$$x[n] \overset{h}{\rightarrow} y[n] \therefore L_D y[n] \rightarrow y'[n] \tag{2.3a}$$

$$L_D x[n] \overset{h}{\rightarrow} y''[n] \tag{2.3b}$$

$$y'[n] = y''[n] \tag{2.3c}$$

Our interest in LTI systems is that they are the building blocks of a CNN that will be shown in the next section. Additionally, the contribution of this work heavily depends on the LTI filters, as it is detailed in Chapter 4.

2.2 GENETIC ALGORITHM

GA is an optimization and search technique introduced by Holland (1975) inspired in the principles of natural selection proposed by *Charles Darwin* (HAUPT; HAUPT, 2004). It comprises generating a population of individuals (temporary solutions) that are evaluated

by genetic operators to find an optimized solution for a given problem. A GA usually comprises the following steps:

1. Fitness function definition: It is necessary to define a mathematical function used to verify the suitability of a solution for a given problem. The best solutions have the greatest fitness. In this work, the fitness will be given by a function that indicates how suitable a filter is to a given problem.

2. Variables definition: In this step, we define the parameters that need to be optimized. In our case, the parameters are the coefficients of an LTI filter.

3. Encoding definition: A GA does not operate directly over the parameters that will be optimized. Instead, the parameters are encoded in *chromosomes* that can be submitted to genetic operators. For example, a filter that has M coefficients composed by rational number can be represented by M' *genes*. These *genes* can be binary, integers, or any other representation that is suitable to be submitted to genetic operators. For the case of integer encoding, the one used in this work, $M = M'$.

4. Initial population generation: GA belongs to the class of population-based metaheuristic methods. It used a set of candidate solutions, the population, to find the optimized solution for the problem. The first generation of the population is usually initialized with random individuals, that eventually evolve to more suitable solutions to the problem.

5. Genetic operators: They are the core of the GA and what helps the system to provide better solutions over the new generation. The key idea of the genetic operators comprises selecting the best solutions, mating its characteristics, and randomly changing parameters.

    (a) *Selection*: This process tends to choose the individuals with the highest fitness in a population P. This step is probabilistic; thus, there is no guarantee that all the best individuals in the process pass to the next generation. This randomness is useful to eliminate local minima (DEY et al., 2010) in the optimization process, as can be noticed in Figure 2.1.

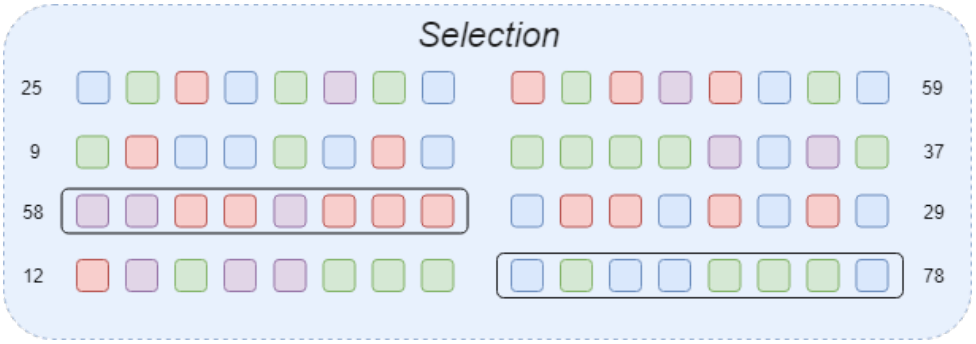


Figure 2.1: *Selection* operator in a population of individuals. The fitness of each candidate is show on theirs side. The selected individuals (inside black rectangles) tends to be those with the highest fitness.

(b) *Crossover*: This is the step that combine characteristics of two individuals. If this combination improves the individual fitness, it tends to pass to the next generation, otherwise it is more likely to be eliminated by the *Selection* operator. The Crossover operator is shown in Figure 2.2. Notice that the *offspring* have a mixed *chromosome* inherited from its *parents*.

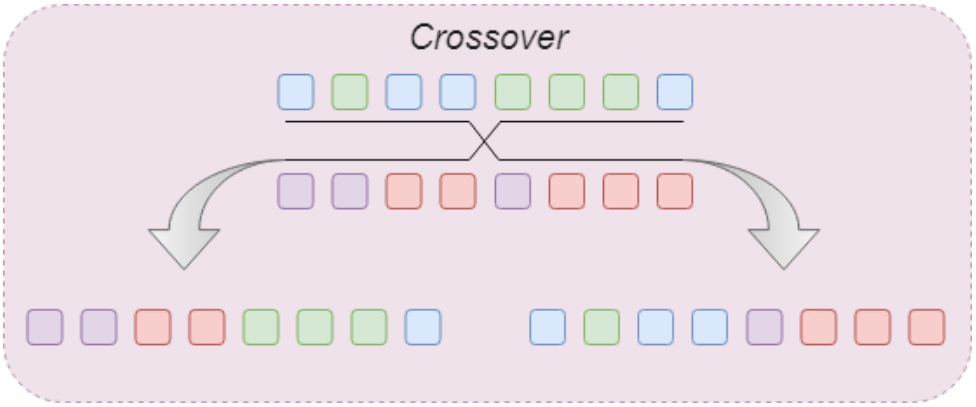


Figure 2.2: *Crossover* operator overview. The *chromosomes* of two the *parents* are combined into the *offspring chromosomes*, who inherit characteristics from both *parents*.

(c) Mutation: It has the objective to insert characteristics that are rarely or even absent from a population. This is done randomly selecting and changing the vale of one (or few) *gene(s)* in a *chromosome*, as shown in Figure 2.3. *Mutation* cannot be overused as this random information can damage the evolved *chromosomes* present in the population.

6. Convergence Check: Once the genetic operators are performed, it is necessary to check the suitability of the found solutions. If none of the solutions satisfy the stopping criteria, the genetic operators are performed again. Otherwise, the GA is

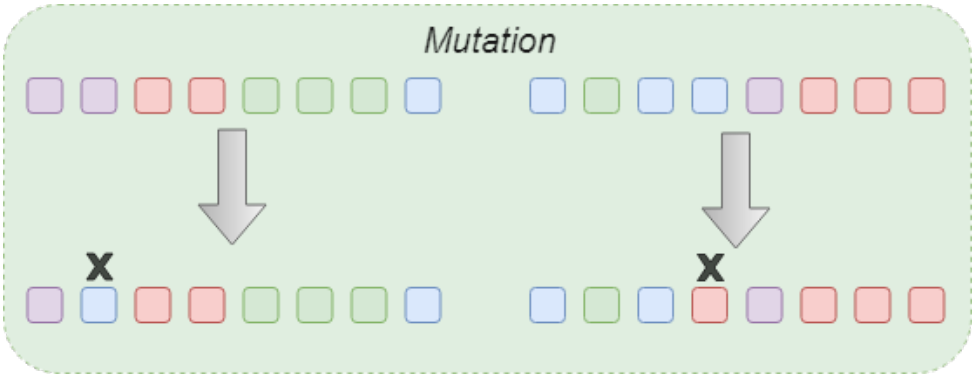


Figure 2.3: *Mutation* operation overview. Notice that some *genes* are randomly changed (black crosses).

halted, and the individual with the best fitness is selected as the solution.

These previous steps are enough to implement a GA for a wide range of problems, including the one proposed in Chapter 4.

2.3 NEURAL NETWORKS

Decision-making processes where there is no well-defined rule make the computational modeling a very tough task. In this case, an artificial intelligence tool that can learn and generalize from examples is a handy instrument to solve problems (RAFIQ et al., 2001). Then, the usage of NNs is a traditional alternative for the solution of numerous problems. A NN consists of a parallel combination of simple processing units which store experimental knowledge and make it available for later use through a training process (HAYKIN, 1994). Notice in Figure 2.4 an example of a topology of a standard NN. If a given input is passed to a NN, an evaluation will be performed through a numerical process, and the network will provide one or more outputs. We present a brief analysis of how a NN performs computation, and we start from its most elementary structure, the neuron.

The basic unity of an NN is a neuron, which is depicted in Figure 2.5. It is composed of inputs, weights, a bias, and an activation function. A neuron is always connected to inputs, and each of these inputs is weighted by a particular value w_x. The neuron adds up all the weighted inputs plus a constant, or bias, as shown in Equation (2.4). After that,

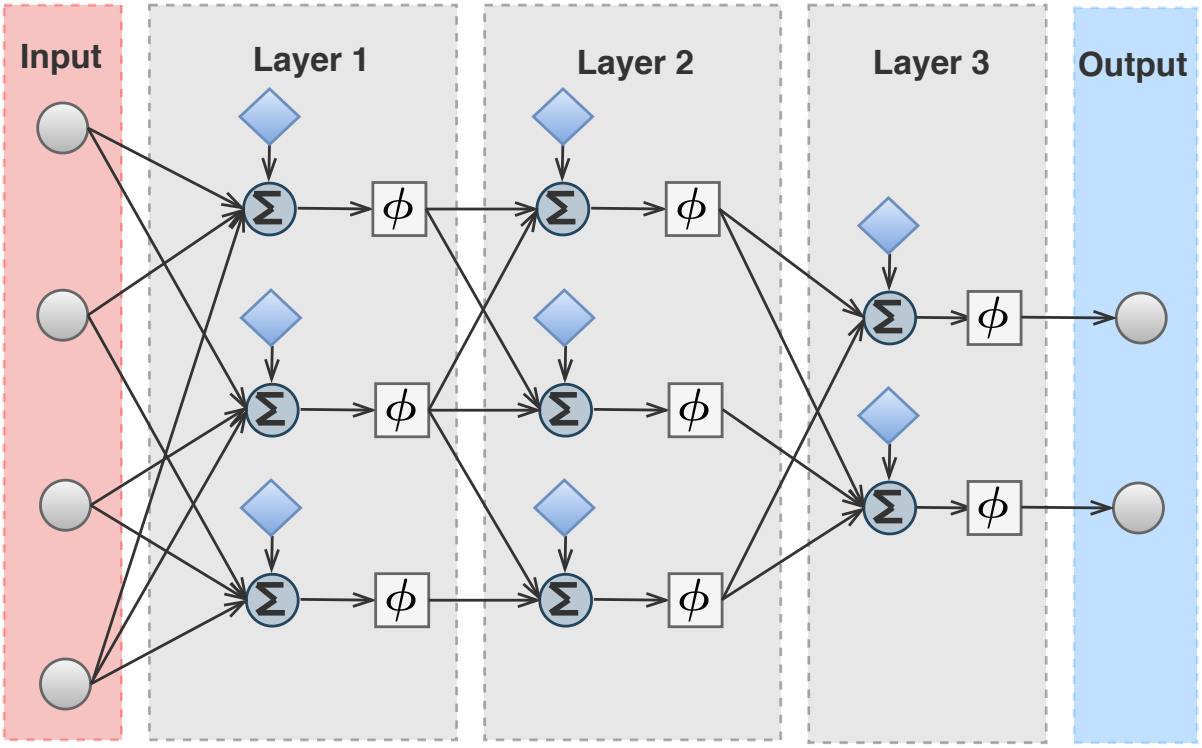


Figure 2.4: Example of an NN. Weighted inputs are added to constants (bias) and then, evaluated by an activation function Φ.

the result of the sum is evaluated by an activation function, as shown in Equation (2.5).

$$z = \left(\sum_{x=1}^{W} w_x a_x \right) + b, \tag{2.4}$$

where a_x is the x^{th} input of the neuron and w_x is the x^{th} weight of the respective neuron, b is the bias.

$$g = \phi(z), \tag{2.5}$$

where g is the output of the value z evaluated by an activation function.

If a NN is required to process intricate patterns, then it must have more than one layer (HAYKIN, 1994) whereas these layers are usually composed of neurons arranged in parallel as seen in Figure 2.4. Multilayers NNs must have a non-linear activation function ϕ; otherwise, one could find an equivalent single-layer NN that would have the same input-output relation (HAYKIN, 1994). Therefore, many activation functions have been proposed in the literature, but in this work, it is presented only two of them, which are:

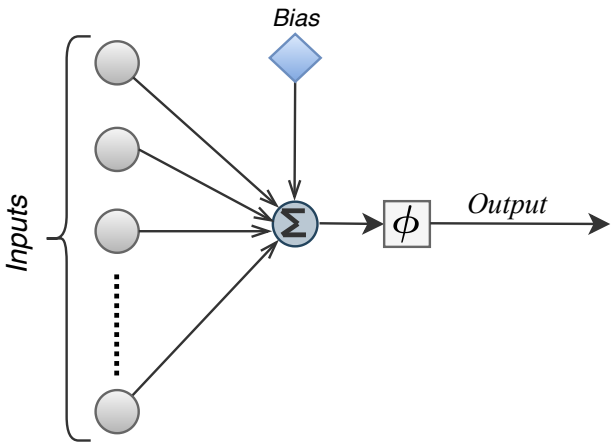


Figure 2.5: Neuron Representation

Sigmoid (Eq. 2.6) and the $ReLU$ (Eq. 2.7) functions.

$$\phi(z) = \frac{1}{1 + e^{-z}} \tag{2.6}$$

$$\phi(z) = max(0, z) \tag{2.7}$$

The $ReLU$ function has been widely used for CNNs. On the other hand, the sigmoid function is a classical function for fully-connected NN and also has been used in the last layers of some CNNs.

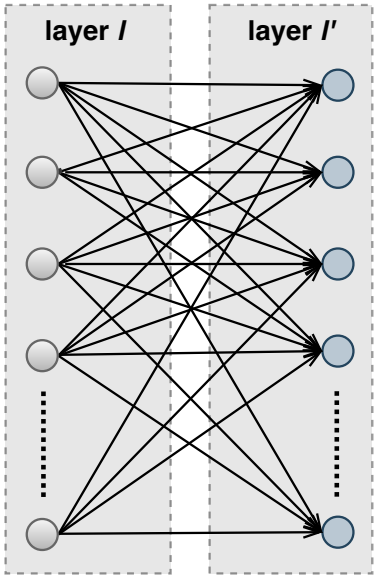


Figure 2.6: Fully-connected Layer Representation - Bias and activation function omitted for better visualization.

Finally, it is important to highlight a special topology of a layer in NNs, the fully

connected layer, which can be seen in Figure 2.6. If all the neurons of a given layer l' are connected to all the neurons of the previous layer l, then we refer the layer l' to be a fully-connected layer.

2.3.1 CONVOLUTIONAL NEURAL NETWORKS

A naive approach to extract information from an image using NNs would be using a standard topology composed only by fully-connected layers. However, the evaluation of a regular size image would require a high computational cost; it also would need an excessive amount of data to be trained, and this architecture would not be shifting invariant (CUN et al., 1994). Therefore, an alternative type of NN should be used for image analysis as a CNN. The CNN plays an important role in many sub-areas of CV as object detection, human action recognition, and VOT, and good examples of these applications are presented in the work of Redmon et al. (2016); Bertinetto et al. (2016); Carreira and Zisserman (2017). Instead of connecting all the neurons of the previous layer to all the neurons of the current layer, the weights in a layer of a CNN are grouped locally. These locally connected weights are called filters, and they are conceived to detect local features (Fig. 2.7). A CNN performs the operation in Equation (2.8) to obtain the feature maps based on the previous layer as depicted in Figure 2.7.

Convolutional layer: The basic structure of a CNN is the kernel or filter. This structure needs to be convolved to the feature map of the previous layer to provide an output, as shown in Figure 2.7. The filters of a CNN are specialized in extracting specific characteristics from an input feature map. In a deep CNN, the filters in the first layer extract low-level features from the signal, and these filters resemble traditionally handcrafted filters, as Gabor filter and edge detectors. Deep layers filters are specialized to extract high-level information related to the type of application the CNN was trained for, as described by Nanni et al. (2017); LeCun et al. (2015); Pouyanfar et al. (2018). One should notice that these filters are LTI, that follow the properties discussed in Subsections 2.1.1 and 2.1.2 are learned according to a given dataset distribution. Therefore, the shape of the filters depends on the data provided during training time.

Notice in Figure 2.7 how $K_{(l')}$ convolutional filters of order $M_l \times N_l$ operate over an input feature map at layer l which has K_l channels of dimension $U_l \times V_l$ that results in

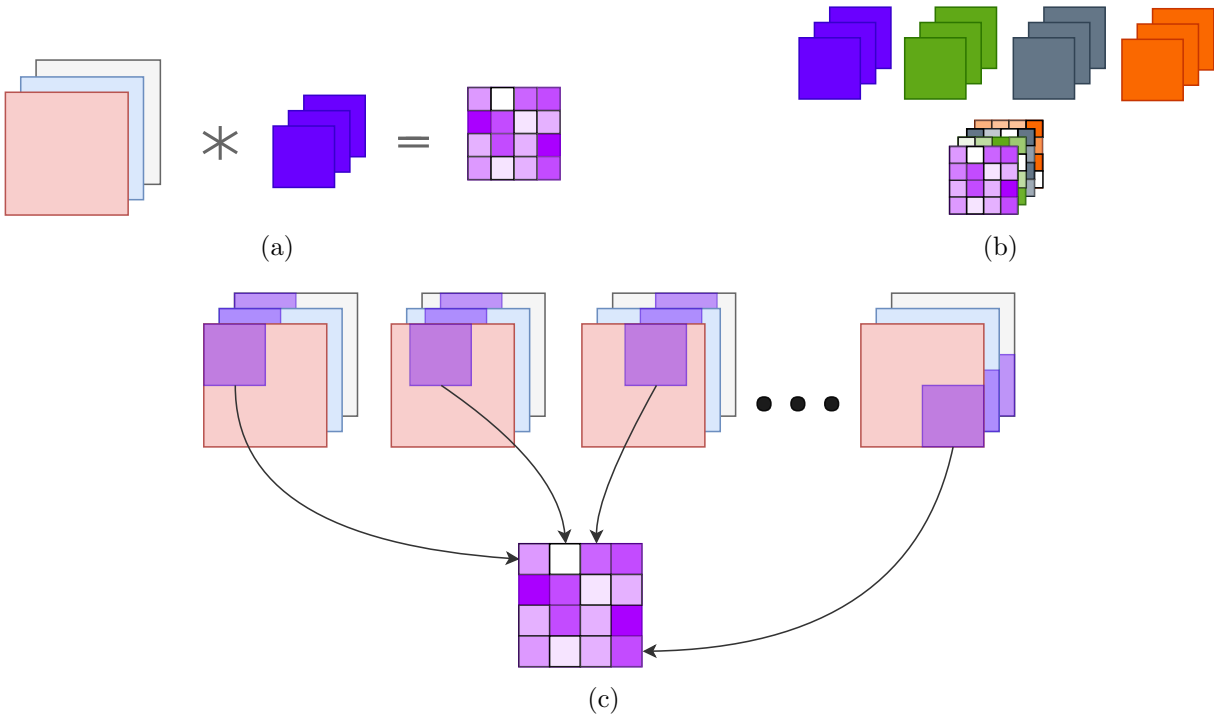


Figure 2.7: How the convolutional filters work (a) An input feature map convolved with a filter. The convolution of both results in an output feature map. (b) Feature map obtained by the evaluation of the four filters over the previous feature map (c) Illustration of how the feature map is obtained. Simplified visualization of Equation 2.8 and 2.10. Bias is omitted for better visualization.

an output feature map in layer l' of $K_{l'}$ channels. This operation can be mathematically described in Equation (2.8), which is the general formula for a generic layer in a CNN.

$$z_{l'}^{k_{l'}}[x_l, y_l] = \left(\sum_{k_l, m_l, n_l} w_{l,k_{l'}}^{k_l}[m_l, n_l] \cdot a_l^{k_l}[x_l \cdot s_l + m_l, y_l \cdot s_l + n_l] \right) + b_l^{k_{l'}}, \tag{2.8}$$

where the domain D of $z_{l'}^{k_{l'}}(x_l, y_l)$ is given as follows:

$$D = \{x_l, y_l \in \mathbb{N} \mid 0 \leq x_l \leq U_l - M_l \; and \;\; 0 \leq y_l \leq V_l - N_l\}, \tag{2.9}$$

Notice in Equation 2.8 that $w_{l,k_{l'}}^{k_l}$ represents the ${k_{l'}}^{th}$ filter at layer l and k_l represents one of the K_l channels of the filter; $a_l^{k_l}$ represents the feature map at layer l which is composed by K_l channels; $b_l^{k_{l'}}$ is the bias for ${k_{l'}}^{th}$ filter in layer l; $z_{l'}^{k_{l'}}$ is the feature map, before the evaluation of an activation function at the next layer l' that has $k_{l'}$ channels; finally s_l represents a sub-sampling parameter called *stride* at layer l. The result of Equation 2.8 is then evaluated by an activation function $\phi(\cdot)$, that provides an output

feature map is given by:

$$a_{l'}^{k_{l'}}[x_l, y_l] = \phi\left(z_{l'}^{k_{l'}}[x_l, y_l]\right), \tag{2.10}$$

where $a_{(l')}^{k_{(l')}}$ is the feature map in the layer l', that has $k_{l'}$ channels.

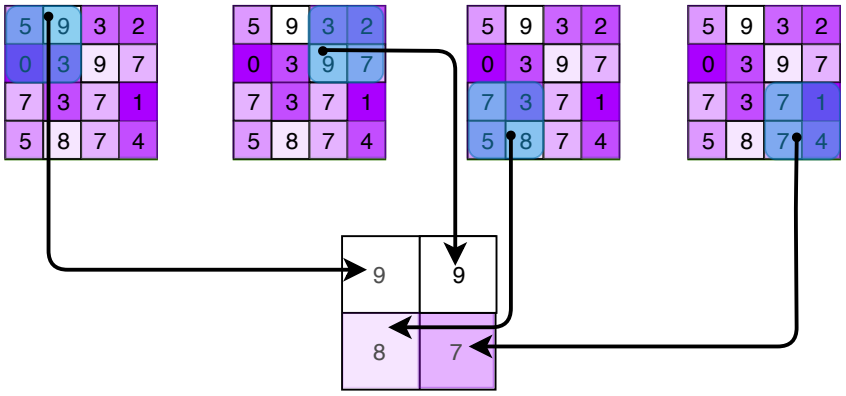


Figure 2.8: An example of a *max pooling* operation being performed in a layer of the feature map shown in Figure 2.7. Notice that the pooling neighborhood is 2×2 windows, in blue, and the stride s is also 2, what results in a feature map of size 2×2. The result of the operation is observed at the bottom of the figure, where whiter values represent higher values in the feature map.

Pooling layer: Essentially, pooling is a process that summarizes the presence of a feature in a channel k, of a neighborhood $N \times N$, for a given a feature map a (KRIZHEVSKY et al., 2012). It turns out that the precise position of each feature is not relevant for identifying patterns by the convolutional filters. Alternatively, even worse, it can also become harmful for feature identification by the network because of the spatial variability of the features in an input image (LECUN et al., 1998). Then, one way to overcome this issue is performing sub-sampling over the feature map while keeping the information provided by the layer. It is where the pooling operation can be helpful.

Pooling layer generates another map that indicates the presence of a feature in a region in $N \times N$ according to an operation, as maximum presence (max pooling) or the average presence (average pooling) of a feature in an $N \times N$ region. However, the displacement of the sampling region $N \times N$ over the feature map does not have to be unitary, but it can skip s unities on each dimension as shown in Figure 2.8. This displacement is the *stride* and this is what promotes the sub-sampling in a feature map. Then, sub-sampling is associated with pooling preserves the presence of the features in the feature map. Notice

that a pooling layer does not have weights, and because of that, it is not unusual to refer to a pooling layer as part of the previous convolutional layer. An example of pooling over a channel of a feature map is shown in Figure 2.9.

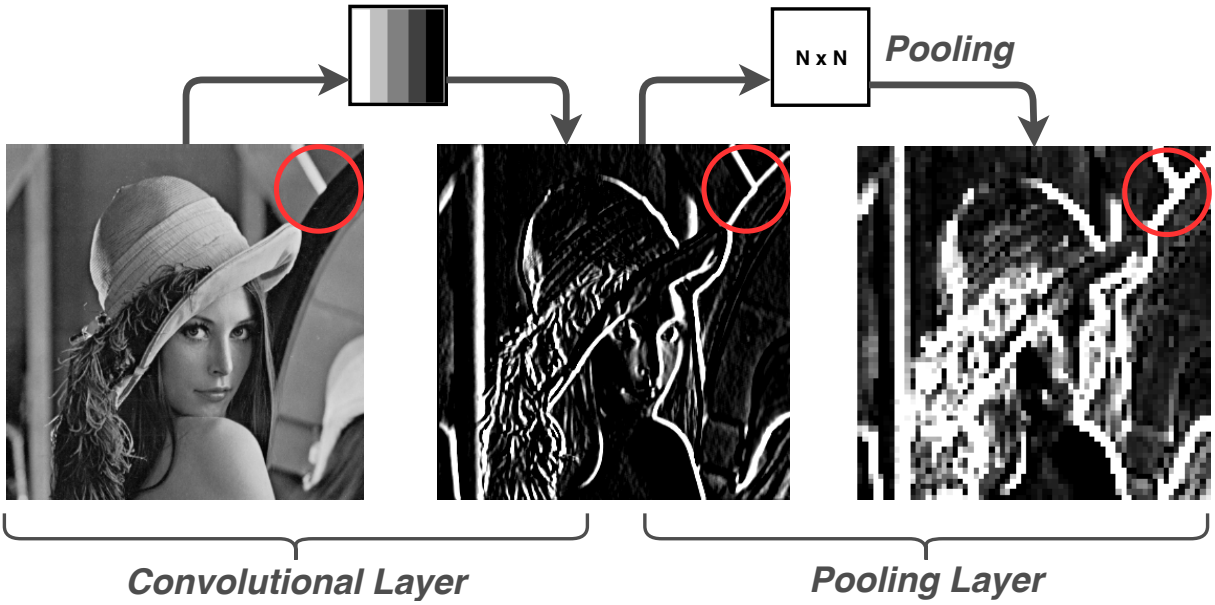


Figure 2.9: Overview of the evaluation of a mono-channel image by hypothetical filter followed by the *max pooling* operation. Notice at the top-right corner, in the red circle, that the filter highlights borders in the image. *Filters and images are not in scale.*

2.3.2 FULLY CONVOLUTIONAL NEURAL NETWORKS

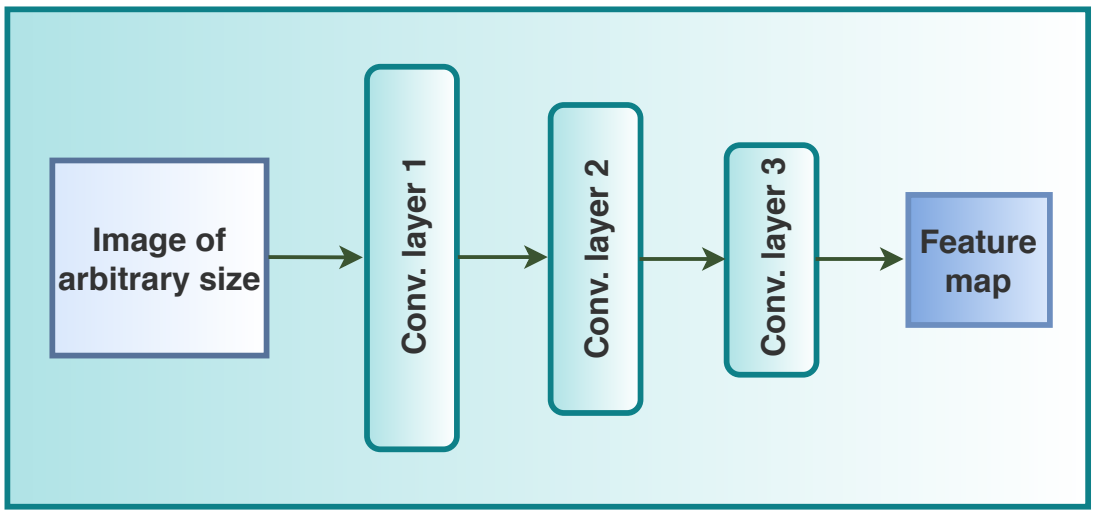
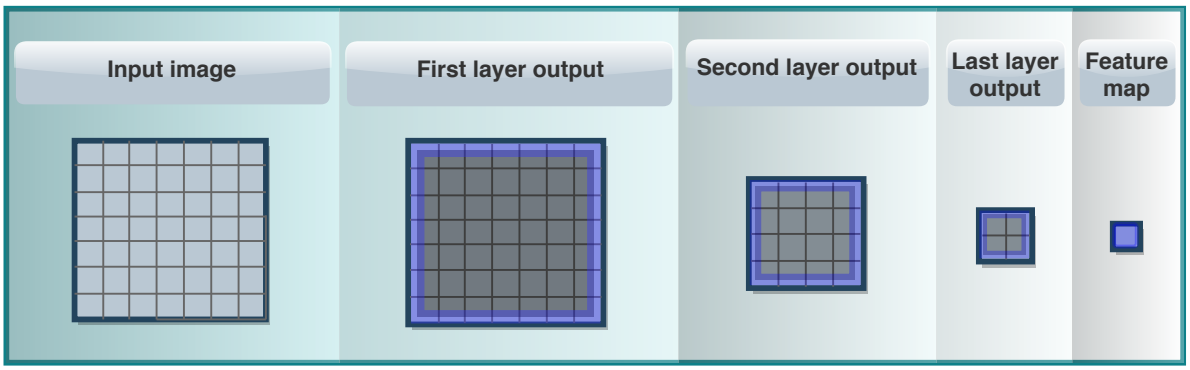


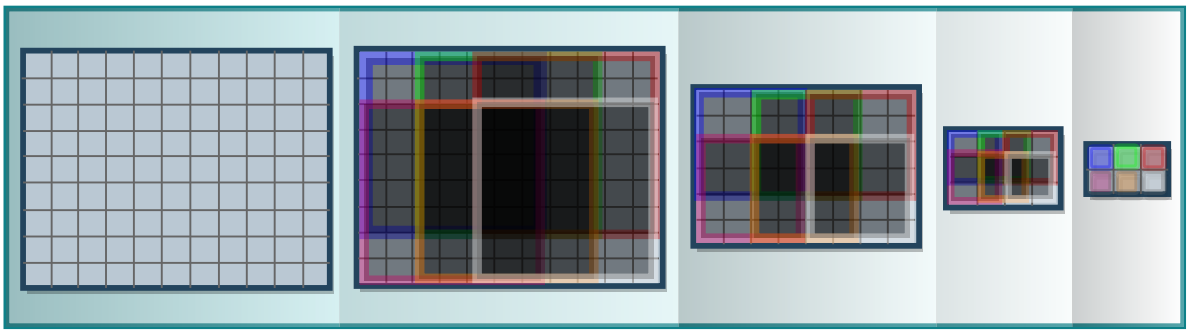
Figure 2.10: Overview of generic FCNN's topology composed of 3 layers. Notice that this network does not present fully-connected layers. This topology allows evaluation of images of different sizes.

A Fully Convolutional Neural Network (FCNN) does not present fully-connected layers

in its topology, as shown in Figure 2.10. Networks with this topology have some interesting characteristics as discussed as following.



(a)



(b)

Figure 2.11: Evaluation process of an FCNN (a) Evaluation of an image of atomic size. It outputs feature map is reduced to a single feature vector. In this case, the output resembles the evaluation of an ordinary CNN. (b) Evaluation of an image greater than atomic image. In this case, there is a feature map whose size is given by the equation 2.11.

1. Variable input size: An FCNN can handle images of different sizes with no need for resizing the input. However, the network does not provide a single output value, but rather an output map of values a, or *feature map*, whose size is proportional to the input image size. There is no theoretical upper limit for the network's input size, but it is not possible to evaluate an image whose size is below a lower limit. We are going to refer to this lower limit size as *atomic size* and an image patch that has this size as *atomic patch*. In the case of the atomic patch, the feature map is composed of a single feature vector, as shown in Figure 2.11 (a). For images greater than the *atomic patch*, each point on the feature map is associated with a region of the input image with *atomic size*, as can be observed in Figure 2.11 (b). The size

of the feature map is given in Equation (2.11), where for the sake of the simplicity, it is analyzed only one of the two dimensions of the feature map.

$$d = \frac{N - \gamma}{s} + 1, \tag{2.11}$$

where, the dimension size d, such that $d \in \mathbb{N}^*$, of the feature map is proportional to the dimension x of the input image, whose size is N and the stride s over the input image. The input image must have at least the atomic size γ. Notice that the term $\frac{N-\gamma}{s}$ must be a natural number; otherwise, it would violate the translation invariance, which is the property that will be discussed as follows.

2. Translation invariance: This property guarantees that an FCNN outputs the same descriptor on the feature map for a given atomic patch wherever it is located in the input image. Bertinetto et al. (2016) define this property as follows:

$$a(L_{s\tau}\boldsymbol{I}) = L_{\tau}a(\boldsymbol{I}), \tag{2.12}$$

where $a(\cdot)$ represents the output of the last feature map of a FCNN, s is the stride and L represents the translation operator of a displacement τ and $\boldsymbol{I}$ is the input image.

3. Computational performance: The topology of an FCNN allows parameters sharing during the evaluation process of an image, and the output map is obtained in a single evaluation. This process is equivalent to evaluate each atomic patch to compose a feature map a (LONG et al., 2015). However, an FCNN is significantly faster to compose a feature map than a naive evaluation of each single atomic patches by an ordinary CNN.

2.3.3 SIMILARITY COMPARISON

One way to compute the similarity between two images, $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$, is to map them into feature vectors $\boldsymbol{z_1}$ and $\boldsymbol{z_2}$, respectively, that represent these images in a given feature space as shown in Figure 2.12. Then, it is possible to apply a metric to measure the similarity between these encoding vectors. It is formalized as follows:

An encoding function $f : \mathbb{R}^2 \rightarrow \mathbb{R}^K$ is a transformation that maps an input image $\boldsymbol{I} \in \mathbb{R}^2 \rightarrow \mathbb{R}^C$ into a feature vector $\mathbf{z} \in \mathbb{R}^K$ that is defined by:

$$\mathbf{z} = f(\boldsymbol{I}). \tag{2.13}$$

The similarity S between two images, $\boldsymbol{I_1}$ and $\boldsymbol{I_2}$, encoded by the vectors $\mathbf{z}_1$ and $\mathbf{z}_2$, respectively, is obtained through a generic similarity function $l : \mathbb{R}^K \rightarrow \mathbb{R}$ represented by:

$$S = l(\mathbf{z}_1, \mathbf{z}_2), \tag{2.14}$$

where low values of S means high similarity and high values of S means high dissimilarity. The similarity function l can be any similarity metric. One of the most straightforward similarity metrics is the Euclidean distance. For example, assuming the Euclidean distance as the similarity metric between two images, notice in the example of Figure 2.12 that similar images are placed close together in the given features space while dissimilar images are localized far apart from each other.

2.3.4 SIAMESE NEURAL NETWORKS

A Siamese Neural Network (BALDI; CHAUVIN, 1993; BROMLEY et al., 1994) is a type of NN dedicated to comparing the similarity between inputs, whereas it can generate a

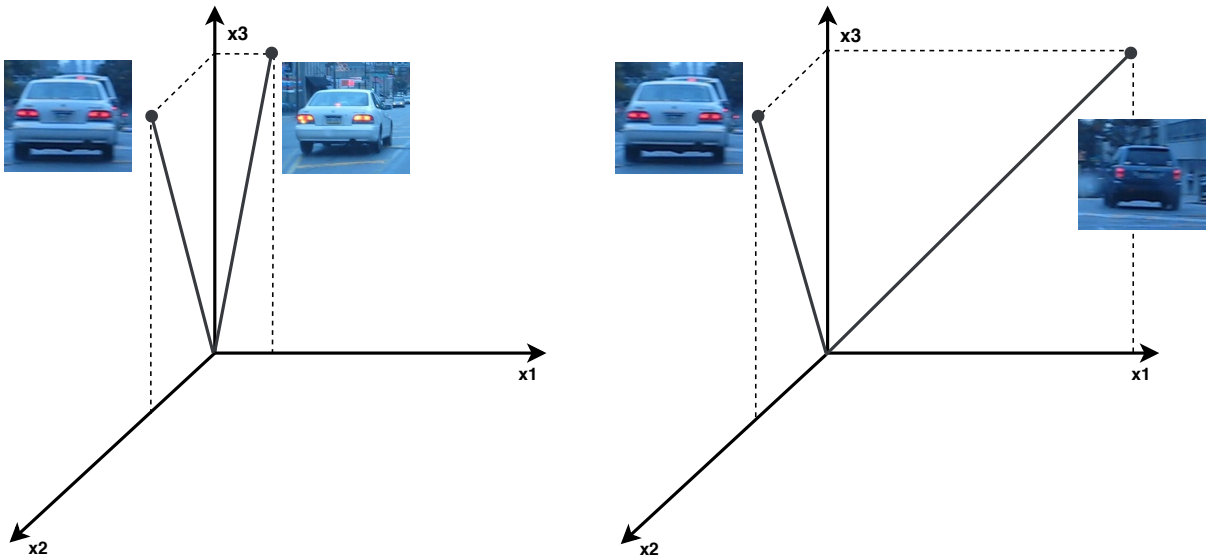


Figure 2.12: Points encoding the respective images in a 3-dimensional space. Notice that very similar images are placed close from each other, and different ones are localized far apart.

similarity ranking between given inputs (KOCH et al., 2015). It embeds an encoding function (Eq. 2.13) and it usually includes a similarity metric function (Eq. 2.14) to evaluate the resemblance between two inputs. The topology of a SNN typically consists of two parts:

1. Twin Networks: These are sub-networks with identical topology, which usually share the same weights. Each one evaluates one of the two inputs that generate the respective feature vectors. The twin networks play the role of the encoding function in Equation (2.13). If these NN are identical with each other, a valuable property mentioned by Koch et al. (2015) must be stated: *two similar inputs cannot be mapped into a very different location in feature space.* However, notice that the previous statement does not hold in the case of twin networks which do not share the same weights, whereas the case of the SNN presented in the work of Vo and Hays (2016). For this case, since the input images are encoded by two different encoding function (Eq. 2.13), two very similar images can be mapped far from each other into the feature space.

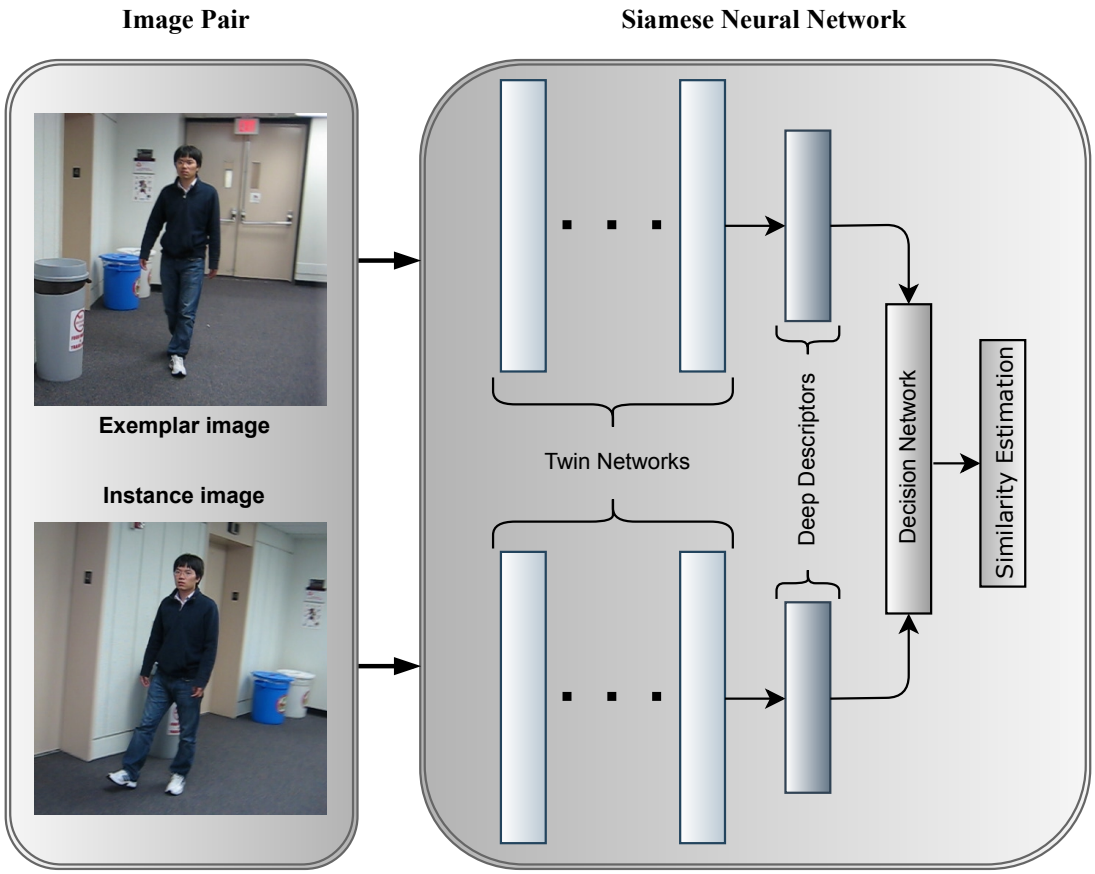


Figure 2.13: Siamese Neural Network overview.

2. Decision Network: It is the sub-network on top of the twin networks. It takes the descriptors generated by the twin networks and computes its similarity. The decision network plays the role of the similarity function (Eq. 2.14). The decision network can be implemented in different ways according to the chosen architecture. The work of Zagoruyko and Komodakis (2015) uses fully connected layers, pooling, and the $ReLU$ activation function as the Decision Networks to output the similarity between patterns. On the other hand, Bertinetto et al. (2016) implemented a simple cross-correlation layer as the decision sub-network to estimate similarity.

A SNN can be used to estimate similarity among different forms of signals as images, sound, biological (MANOCHA et al., 2018; PATANE; KWIATKOWSKA, 2018). However, this work focus on image similarity. Thus, from now on, we restrict our analysis for images. We show the basic structure of a generic SNN for image similarity in Figure 2.13. The twin networks evaluate the inputs and generate the deep descriptor as the output. Then, the decision network evaluates the deep descriptors and outputs the similarity between them.

2.3.5 THE SIAMFC NETWORK

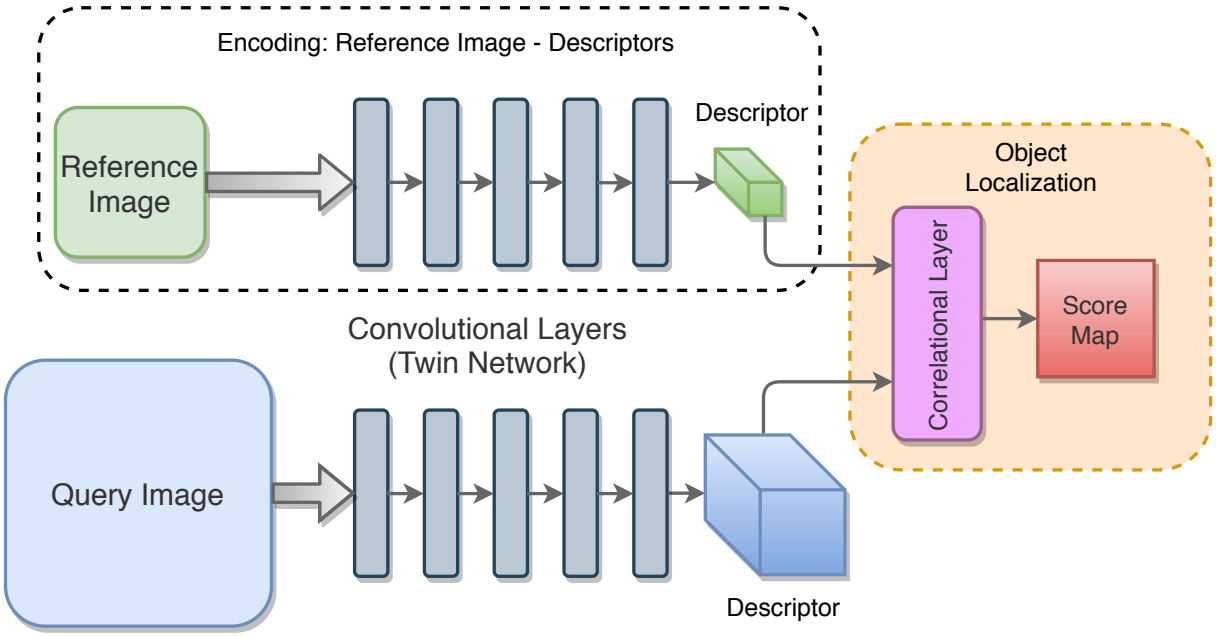


Figure 2.14: SiamFC overview. Notice the proportionality of respective input images and the feature maps.

The SiamFC is a fully-convolutional SNN elaborated for VOT proposed by Bertinetto

et al. (2016), whose twin networks are inspired by the *AlexNet* architecture. The overview of SiamFC topology is shown in Figure 2.14. The twin sub-networks in SiamFC have two mirrored branches and they are composed of 5 convolutional layers and 2 pooling layers. These last ones are located just after the first and the second convolutional layer. An encoding function is then performed by these layers to generate descriptors from images. Additionally, the decision network is composed of a single layer network that applies the cross-correlation between the exemplar and instance feature maps. This layer performs an operation similar to the cross-correlation widely used in DSP Theory, where Orfanidis (1995) describes a more detailed explanation about it. However, as there are K features maps, all the maps are added up. Therefore, the result of the correlation is used to create a score map that indicates the likelihood of the object in the reference image is found in a given region of the candidate image. Each atomic patch, of size 87×87 pixels, outputs a deep descriptor of $K = 256$ dimensions. There is a stride of 8 pixels for both input images. Then, the feature maps dimensions are given in Equation 2.11. Originally, this network uses a reference image whose size is 127×127 pixels, and a candidate image of size 255×255 pixels. Then, recalling Equation (2.11), the outputs are feature maps of dimension $6 \times 6 \times 256$ and $22 \times 22 \times 256$. The correlation of the final feature maps of the input images results in a 17×17 score-map.

In summary, the SiamFC stands out in the literature not only because of its performance and the fact it requires a relatively low computational cost, but because it also presents the handy fully-convolutional property. Therefore, these are the reasons that justify the usage of this architecture in our work.

3 LITERATURE REVIEW

This chapter presents a brief review of the significant works in VOT literature and helps the reader to follow a historical context to understand how frameworks for VOT have evolved. We discuss the advantages and drawbacks of the current approaches for VOT, and it helps the reader understand the choice of an SNN as the framework for our tracking method in comparison to alternative approaches.

3.1 THE CLASSICAL APPROACHES TO TRACKING

The first proposals for VOT were simple template-based methods that relied on Normalized Cross Correlation (NCC) to match a given pattern from incoming frames (TAO et al., 2016). The VOT approaches naturally evolved to more sophisticated models, which were more appropriate to deal with adverse situations as noise, illumination change, articulated object, occlusion, and others. In the first half of this decade, those trackers that were well established in CV community used very sophisticated frameworks to overcome a variate of situations (as the ones mentioned previously) that could lead to tracker failure.

A traditional example of the classical tracker was the Tracking-Learning-Detection (TLD) framework, proposed by Kalal et al. (2012). This tracker was able to learn different appearances of the object and the background through a semi-supervised learning procedure. This process was performed online, and it updated an object detector that indicates the Region of Interest (RoI).

The Kernelized Correlation Filter (KCF) tracker was another successful example. It introduced the usage of correlation filters for tracking proposed by (HENRIQUES et al., 2014). This method consists mainly in finding a transformation over feature maps that would minimize the error between the predicted output and the Ground-truth (GT), on an online process. This method and achieved noticeable performance in the literature, although it would require a high computational cost (it needed a continuous online computation for weight update).

Another remarkable tracker was the Appearance Learning In Evidential Nuisance (ALIEN) (PERNICI; BIMBO, 2014). It relied on the Scale Invariant Feature Transform (SIFT) to extract and obtain the feature representation of an object to discriminate

it from the background. It also used an algorithm intended to avoid false information when the RoI was occluded. ALIEN presented a very competitive performance among classical approaches.

Although these trackers could have reasonable performance, they heavily relied on an online process, with few, or even no, *a priori* information about the object intended to be tracked. Also, these methods could not extract semantic information from the objects (HELD et al., 2016; NING et al., 2017). In contrast, recent approaches for VOT were able to extract more discriminative information about the object due to the usage of DNN. When compared to the historical trend, some approaches presented in the last few years rely on simplistic tracking algorithms. These frameworks achieved state-of-the-art performance, as the SiamFC and Generic Object Tracking Using Regression Networks (GOTURN). These top-performance methods shared one point in common: they are based on DL (LI et al., 2018).

3.2 DEEP LEARNING

DL revolutionized ML area in the last years. This revolution came from the successful usage of deep methods in a huge variety of applications as speech recognition, natural language processing, genomics, drug discovery, biomedicine, information retrieval, CV, and many others (POUYANFAR et al., 2018; LECUN et al., 2015). Some authors explained the reason for the widespread adoption of DL by the scientific community. LeCun et al. (2015) highlighted that a key advantage of DL methods is that they are capable of learning useful discriminative features from data distribution in an automatic way with no need to be designed by experts. Nanni et al. (2017) attributed the successful performance of DL because deep models can take into account the semantic relationship between a given data distribution. Then, DL models can provide a more discriminative capability for deep descriptors when compared to handcrafted models. All in all, DL enabled fast advances in many areas of science.

The breakthrough for the adoption of deep models by CV community dates from 2012. In that year, the *AlexNet* Neural Network (KRIZHEVSKY et al., 2012) was introduced. It was the first deep architecture that presented a very significant improvement over traditional methods in image classification. After that, other deep architectures were proposed, as VGG (SIMONYAN; ZISSERMAN, 2014) and Inception (SZEGEDY et al.,

2015) which successfully improved performance in the supervised image classification area. According to Pouyanfar et al. (2018), the problem was regarded as ''solved'' for this area. The success in image classification inspired the development of solutions based on DL in many other CV applications as semantic segmentation, object localization, video analytics. They all achieved a new stage of advancement due to DNN (POUYANFAR et al., 2018). Therefore, following the same trend, VOT was widely benefited by DL as well.

3.3 DEEP LEARNING FOR TRACKING

Numerous DL trackers were proposed in the literature since the middle of this decade. Many of these deep trackers have one common point: they apply feature extraction using a DNN.

The findings of Wang et al. (2015) can reason a justification for the strong predominance of DL methods in tracking. In their work, they performed a useful analysis of the typical components of a tracker. Then, they placed experiments to find what part impacted the most in a VOT system. They figured out that feature extraction (the conversion of the raw image into an informative representation) was the most critical part of a VOT framework. Furthermore, it was reasonable to expect that deep trackers could present an improved performance when compared to traditional approaches. It was because the deep representation of features had already outperformed traditional feature descriptors in many applications (ZAGORUYKO; KOMODAKIS, 2015; NING et al., 2017; NANNI et al., 2017).

Thus, the natural consequence of the superior performance of DL for feature extraction was the Deep Learning Tracker (DLT) method proposed by Wang and Yeung (2013). This tracker was the first in the literature to apply a deep neural network to tackle the VOT problem (HUANG, 2017). By the time of the proposal, the authors also realized that deep features would bring a significant improvement of a tracker. Then, in this novel approach, the DLT did not use feature extraction based on traditional methods, but instead, a DNN was implemented to provide a feature representation of the object intended to be tracked. On top of the deep descriptors, it uses a particle filter to obtain the final output. Although this framework was not an end-to-end deep learning approach, it was the accurate indication that DL could successfully be applied to VOT.

Many other proposals followed the DLT. GOTURN was a very successful end-to-end

framework for VOT that was proposed by Held et al. (2016). It consisted of a deep neural network that outputs a BB for each frame based on the input of two images. During training time, it learns a generic relationship between motion and object appearance that is used at running time. A significant advantage of this tracker was that it did not have an online model to update, what contributed for achieving satisfactory tracking performance at more than 100 Frames Per Second (FPS) by the time the work was proposed.Similarly, Ning et al. (2017) presented an end-to-end approach that also extracted motion information in a tracker. They proposed an architecture that combined Recurrent Neural Network (RNN) on top of an object detection CNN. This network presented an excellent performance under severe occlusion as it explores the historical motion of the object through the video. Although these two previous methods emphasize temporal information, they did not present a superior performance when compared to state-of-the-art methods. According to Huang (2017), trackers which would explore time component more appropriately is an open-end topic in VOT community.

Recently, several methods were also based on correlation filters for tracking inspired by the KCF framework. Danelljan et al. (2016) proposed the Continuous Convolution Operator Tracker (C-COT). In this proposal, it was used continuous correlation filters in multi-resolution feature maps of a DNN. This continuous representation of the feature map was important because different convolutional layers presented the different spatial resolution. Although the last layer of CNNs would present features that contains structural and spatial information of the tracked object, it also would present a lower resolution. In contrast, the first layer presented a high resolution that could be complementary to the last one. As the requirement of a correlation filter is the same resolution for all the feature maps, the major contribution of this work was the proposed transformation of feature maps to a continuous spatial domain through an interpolation process. After that, all the maps could be re-sampled by the same resolution, meeting the criteria for correlation filters usage.

Danelljan et al. (2017) improved the previous proposal introducing the Efficient Convolution Operators (ECO). They reduced the number of filters for the convolutional layers of the C-COT approach. It was done by introducing a linear combination of the convolution filters in C-COT, which was necessary for decreasing the number of parameters to be learned. It turned out that this combination not only reduced the computational

burden of parameters but also increased tracking performance according to the authors. Another contribution of their work was a model update strategy that groups new samples to compose the object model. This proposal led to a decrease of redundancy and increased the variability of the object model.

Another DL tracker is the Correlation Filter Network (CFN). It works with numerous tracking modules based on correlation filters, where each of them covers a specific type of object change, like blurring, structural change, occlusion an others. These modules are validated by a pre-trained NN, the attention network. The modules with the highest score are used to judge the object appearance. Then, the one with the highest score is used to the output the position and scale of the RoI. Once the position is estimated, the remaining tracking modules are updated according to the new information (CHOI et al., 2017).

Finally, a high-performance tracker that uses a multiple front analysis is Multi-Domain Network (MDNet). It relies on a multiple branches layer, where each of these branches performs a binary classification to identify the RoI. On top of it, an online updated layer is used to combine the object representation of such branches, and then, the object can be localized (NAM; HAN, 2016). Although it presents a noticeable performance, this proposal is very slow for online tracking.

3.4 TRACKING WITH A SIAMESE NEURAL NETWORK

The usage of SNNs in CV applications dates from the nineties in fingerprint comparison (BALDI; CHAUVIN, 1993) and in signature verification (CUN et al., 1994). However, it was only recently that SNN was highlighted in the literature. It was because it outperformed traditional methods used in image comparison, as the traditional SIFT descriptor (ZAGORUYKO; KOMODAKIS, 2015). Then, the natural consequence of this high-performance framework in image comparison was the usage of SNNs for VOT problems. Although it is a very recent approach, it already presented a noticeable performance in the latest public evaluations (KRISTAN et al., 2018, 2017). These approaches usually presented a relatively simple framework associated with rapid evaluation and high performance.

The Siamese INstance search Tracker (SINT) was the first tracker that relied on a SNN, and it was proposed by Tao et al. (2016). SINT had a simple framework dedicated

to comparing a reference image patch, the RoI in the first frame, to a candidate image in each frame of the video. In every frame, SINT compares several BBs close to the region where the object was detected in the previous frame. Then, the most similar BB is chosen to be the region where the object is located in the current frame. This tracker had no occlusion detection, no object model updating scheme, no object motion modeling, no geometric metric, no association of trackers and it still reached the state-of-the-art performance by the time the work was published. Notice that even this framework was very simplistic, it outperformed well-established methods as TLD, which has occlusion module detection and object model updating.

After the breakthrough of the first tracking dedicated SNN, several other works were proposed in the literature, aiming real-time tracking. An example was the Discriminant Correlation Filters Network (DCFNet) framework, proposed by Wang et al. (2017), that merged the correlation filters approach to an SNN. The result was an efficient, lightweight system that was able to run up to 65 FPS, according to the authors.

Another successful contribution was given by Bertinetto et al. (2016), who proposed a fully convolutional SNN. The proposed network was known as SiamFC and it is still a state-of-the-art tracker (KRISTAN et al., 2018). Similarly to SINT, it only uses a pre-trained SNN to localize the object during run time. No model update was included. The most significant innovation about this tracker is that it was based on an FCNN, which provided computational advantages over the SINT method. Also, the fully convolutional property of this network made the object localization invariant to the region in an instance image. The SiamFC indicates the confidence of the object to be found in a particular region of the image based on a score-map. Due to its simplicity, performance, and the strong evidence that SNNs are a promising approach for VOT, the SiamFC is used as a baseline to many works to demonstrate new concepts about SNN for trackers. Such concepts includes online weight adaptiveness, correlational filters, rotation invariance, and others (GUO et al., 2017; ROUT et al., 2018; VALMADRE et al., 2017). Therefore, we adopt the SiamFC as the baseline of this work as well. However, for now, we are going to proceed with the presentation of methods that were derived from SiamFC.

Guo et al. (2017) also used a SNN for template matching. Although they had the SiamFC as the baseline of their proposal, they criticized it because of the lack of adaptiveness during tracking time. Then, they proposed the Dynamic Siamese network (DSiam)

architecture, whereas they applied an online transformation on the weights of the original static SNN based on information of previous frames, adding temporal variation to the network. It also learns a background suppression transformation for the candidate image in order to minimize irrelevant background variation.

Rout et al. (2018) presented an improvement for SiamFC providing a rotation-invariant framework. They assume that the rotation of the object from one frame to another was smooth, similarly to the SiamFC assumption for displacement. Then, several images were presented with different rotation angles, and it would compare the similarity between these images, improving its performance in the presence of ration when compared to SiamFC.

3.5 DEEP DESCRIPTORS AND FILTER LEARNING

According to recent works for VOT (VALMADRE et al., 2017; TAO et al., 2016; WANG; YEUNG, 2013; BERTINETTO et al., 2016; DANELLJAN et al., 2016, 2017), it was possible to observe that the representation of objects based on deep descriptors was a common point to those DL methods. In this context, one established approach using deep descriptors was filtering these descriptors using a correlation filter, as discussed in Section 3.3. However, the major drawback of this approach was the intensive computational cost. Additionally, we highlight that these approaches concerned mainly in extracting spatial relation in a feature map of deep descriptors. In contrast, temporal component analysis and filtering is an open-end branch that can be explored (HUANG, 2017).

Recently, Valmadre et al. (2017) used a framework very similar to the one proposed by Bertinetto et al. (2016). They adapted the SiamFC to be used with a correlation filter and provided the system an adaptive method to a temporal weight variation of the feature maps. Instead of having a single feature representation of the target object, based only on the first frame, a new template is computed for each frame. Then, a moving average combines the current template with the previous ones. Although the authors did not explore in depth the temporal combination of templates, this method demonstrates good evidence that temporal filtering the feature representation of the object helps the system performance. Due to this indication, it is reasonable to dedicate more efforts in our work to explore this approach more in-depth, aiming to find a more general weighting of the descriptors. As the moving average can be seen as a particular case of a linear and time-invariant filter, it is possible to look for a more generalized approach to obtain

filters which are more suitable for tracking purposes instead of such simple form discussed above. Thus, one of the motivations of our work is to explore methods to weight the previous descriptor obtained during tracking, as LTI filters, which should, preferably, be designed based on observation the available tracking data.

Then, we explore approaches in the literature that help us to find such filters, or weighting forms, that combine previous templates to give useful information for this work. Additionally, we have to be aware of crucial frames that provide relevant descriptors to the object intended to be tracked, to emphasize the influence of such frames.

However, looking to the DSP literature, we can find some methods to provide similar approaches to our application. Cemes and Ait-Boudaoud (1993) showed a comparison between GA and other methods aimed to find a LTI filter, which showed the suitability of GA compared to other methods. They also suggested a fitness equation that allows us to verify how suitable a given filter is for our application. In a similar approach, Dey et al. (2010) showed the advantage of LTI filter design using GA. Although they warned that the convergence speed and the computational cost is considered a drawback of GA, its capability of easy implementation of parallel processing and strong potential to overcome local minima makes GA a suggestive choice to find a filter. Then, using a GA would give a better response to the tracking problem than a moving average suggested by Valmadre et al. (2017). Ahmad and Antoniou (2006) provided details about the encoding scheme for the genotype, that is useful for the converge of the algorithm which will be used in this work. Altogether, the GA is a reasonable choice to obtain filters that would outperform the usage of the moving average and the traditional implementation of the SiamFC.

4 PROPOSED METHOD

In this chapter, we present the concept of ideal and real SNNs and how it is possible to obtain a signal of descriptors from SNN dedicated to VOT. Following, we propose to complement an SNN to mitigate its limitation, adopting the SiamFC proposed by Bertinetto et al. (2016) as the baseline for our work, although the model can be generalized to other tracking methods based on SNNs. We also present a specific way to complement the SiamFC using the concept of memory blocks of descriptors and how to compute the parameters of each memory block.

4.1 THE IDEAL SNN

An ideal SNN can map different instances of the same object into the same vector in a given feature space. Additionally, different objects cannot be mapped by the same descriptor. Figure 4.1 shows an example of encoding of an object by a SNN.

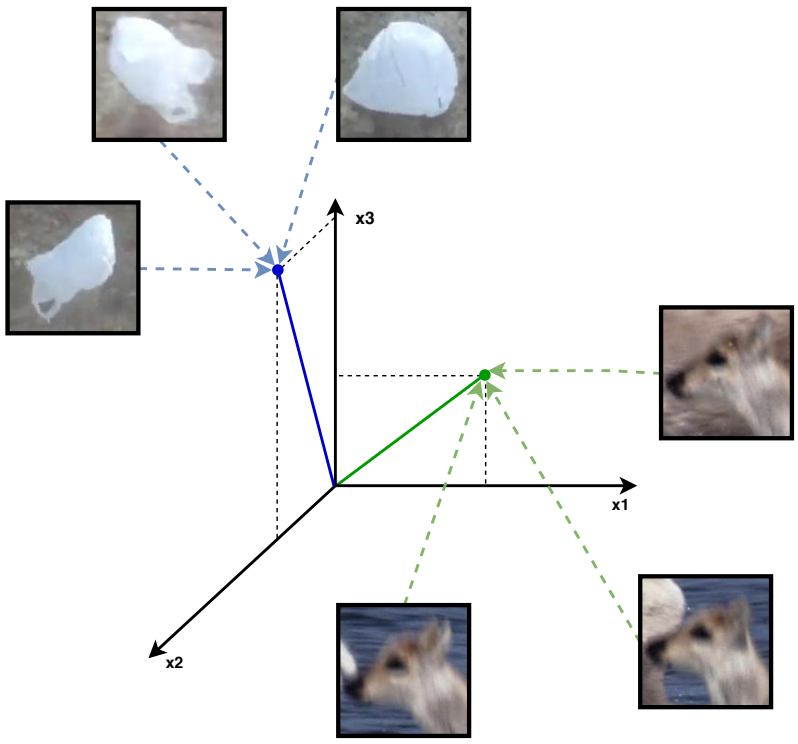


Figure 4.1: Example of object mapping of an ideal SNN into a three-dimensional vector.

In theory, a tracker based on an ideal SNN does not drift from the RoI to a background object, as the single instance of an object is enough to track it in a video with no tracker

failure. However, we do not observe the concept of ideal SNN real systems, and Section 4.2 discuss its limitations.

4.2 THE REAL SNN

A real SNN cannot map different instances of objects into different feature vectors. Therefore, an SNN only ensures that similar images are mapped in the same neighborhood in feature space, as discussed in Subsection 2.3.4. This characteristic makes it possible to mismatch two instances of different objects, and Figure 4.2 shows an example of object mapping from a real SNN. Notice that some instances of different objects can be closer than instances of the same ones.

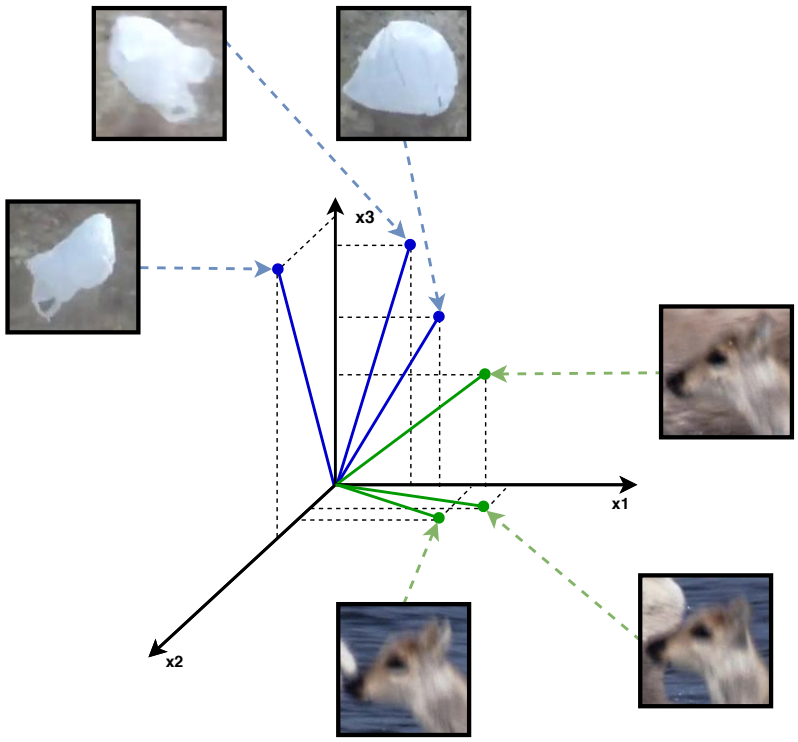


Figure 4.2: Example of object mapping of an ideal SNN into a three-dimensional feature vector.

Although in many real situations a single instance of the object is enough to discriminate the RoI, the mapping limitation of a real SNN no rarely results in tracker failure. Therefore, we propose to extenuate this weakness based on a signal of descriptors presented in the next section.

4.3 SIGNAL OF DESCRIPTORS

Trackers based on SNNs, as SINT and SiamFC, are based only in the descriptor generated in the first frame of a video. This descriptor is the only information that these trackers have to follow the object in a video, as depicted in Figure 4.3.

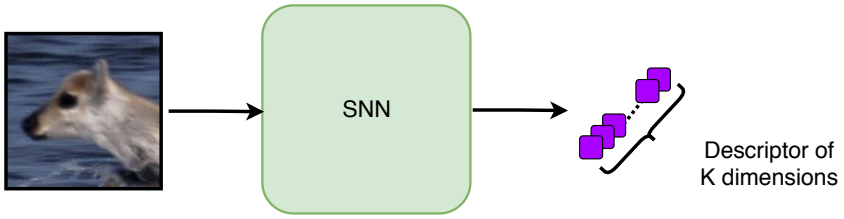


Figure 4.3: Encoding of an image. The single descriptor of K dimensions relies on a single instance of the object and it is all the information that the trackers based on SNNs discussed in Section 3.4 have to perform tracking.

One way to overcome the mapping limitation is by using more than one instance of the same object to compose an improved reference. First, it is necessary to obtain a temporal signal of descriptors instead of using a single descriptor of the RoI. The encoded BBs generate the signal of descriptors outputted by the tracker along with the video, as shown in Figure 4.4:

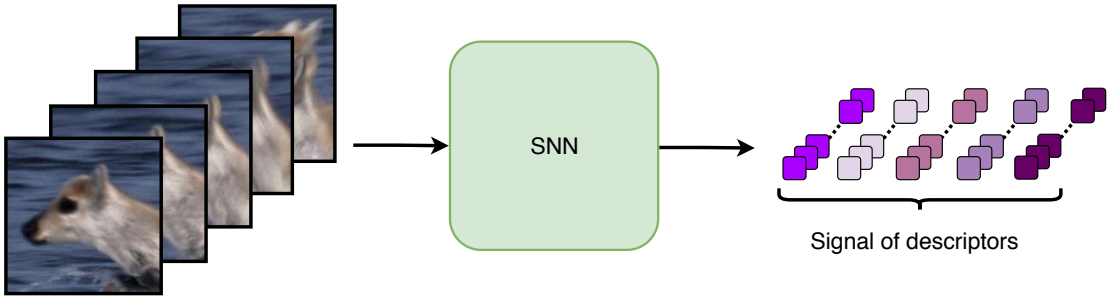


Figure 4.4: Signal of descriptors obtained through an SNN designed for tracking purpose. A signal of descriptors provides a much richer information about the RoI.

Then, a signal of descriptors $\mathbf{z}[n]$ that encodes the BBs given by the tracker, composed by different K-dimensional feature vectors for each frame n is given by:

$$\mathbf{z}[n] = f(BB_n), \tag{4.1}$$

where BB_n is the n^{th} BB which the RoI is believed to be contained in the video and f is the encoding function discussed in Section 2.3.3 whose parameters dependent on the SNN.

Similarly, the signal of descriptors $\mathbf{g}[n]$ that encodes the labeled GT is given by:

$$\mathbf{g}[n] = f(GT_n), \tag{4.2}$$

where GT_n is the n^{th} RoI in a video indicated by the GT.

4.4 THE COMPLEMENTED SIAMFC

Given a signal of descriptors $\mathbf{z}[n]$, it is possible to propose an operation $\mathbb{O}$ to generate a *Combined Descriptor* that better maps the object in the feature space for the frame n. $\mathbb{O}$ can be composed by taking into account all the n BBs outputted by the tracker as shown in the following equation:

$$\mathbf{z}_{comb}[n] = \mathbb{O}(\mathbf{z}[0], \mathbf{z}[1], ..., \mathbf{z}[n-1]), \tag{4.3}$$

where, $\mathbf{z}_{comb}[n]$ is the *Combined Descriptor* signal.

Thus, the goal is to find a combination of BBs' descriptors which improves the performance of the system, as shown in Figure 4.5. Notice that for the case of an FCNN, as the SiamFC, $\mathbf{z}[n]$ and $\mathbf{z}_{comb}[n]$ represent signals of feature maps.

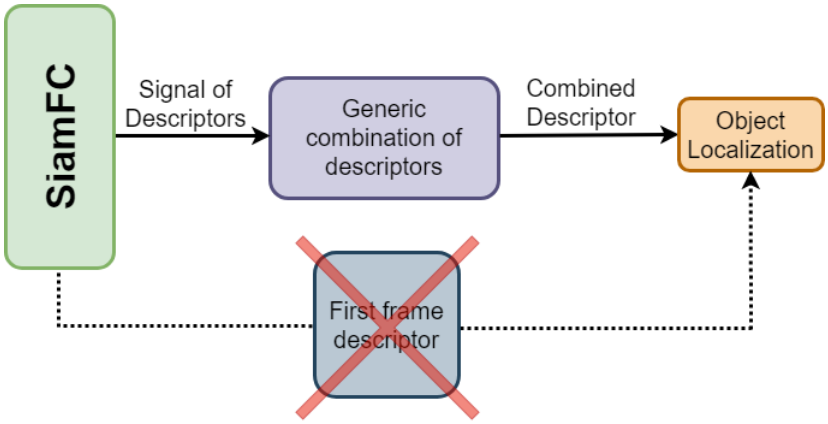


Figure 4.5: Proposal overview

Notice that the $\mathbb{O}$ could be any computation method. For instance, one can train a NN designed to obtain the template in a frame n based in all known object instances present in previous frames. Another approach would be to manually attribute different weights to each of the last descriptors to generate a resultant descriptor to the current frame.

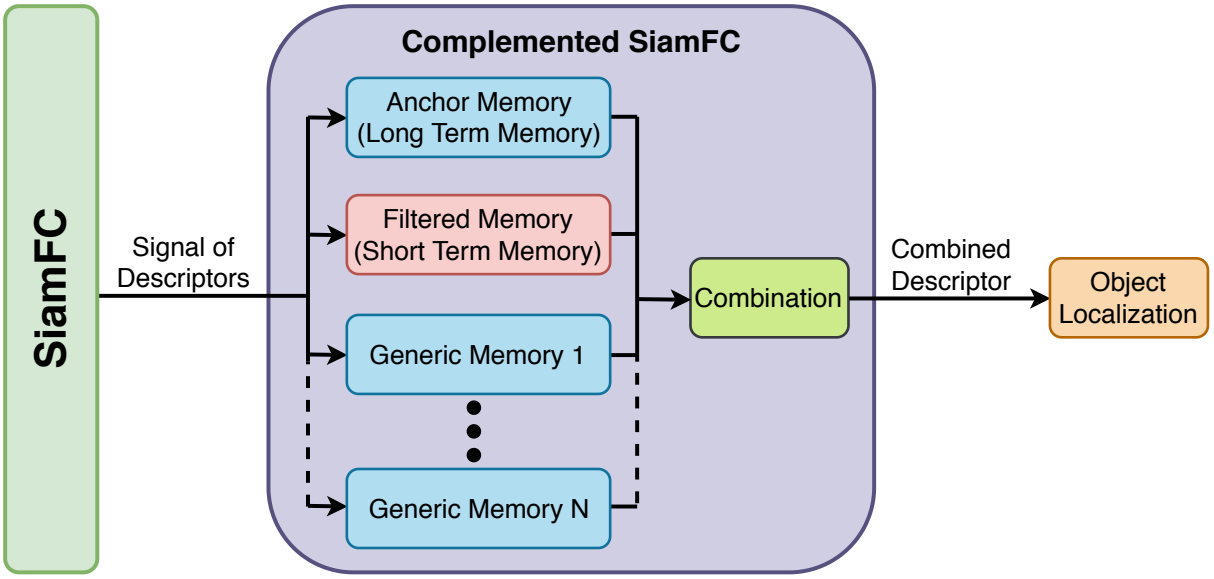


Figure 4.6: Complemented SiamFC based on memory blocks that are combined to generate a Combined Descriptor. Notice that different memory blocks resemble the appearance of the object in different temporal perspectives, these memories generate the Combined Descriptor used to localize the position of the object.

In particular, this proposal restricts to complementing the SiamFC using signals provided by *memory blocks*. We define as *memory block* the combination of descriptors from a limited part of the signal of descriptors. Each of these memories captures different mapping descriptors from the RoI over time, and Figure 4.6 represents this idea. It is possible to use many memory blocks in parallel as *Long Term Memory*, that take into the account information about the earliest descriptors of the RoI; *Short Term Memories*, that combines the descriptors related to the latest appearance of the object; and N number of memories that would capture information about the object in any part of the videos. Notice that memories blocks are also combined through a generic combination to obtain the *Combined Descriptor*.

In this work, we restrict the *Combined descriptor* to be composed only of the *Long Term Memory* and the *Short Term Memory*. The choice for the usage of these blocks relies on some pieces of evidence presented in the literature, described as follows:

1. *Long Term Memory*: Tao et al. (2016) highlights that the only reliable data in VOT is from the first frame. It suggests that a reference about the RoI at the first frames should be kept during the entire process of tracking, avoiding the prevalence of inconsistent information over the reliable one. This information would prevent tracker failure due to drifting, as mentioned by Guo et al. (2017).

2. *Short Term Memory*: Guo et al. (2017) also indicates that a tracker based on the knowledge of a single template BB restricts tracking capability. Additionally, Valmadre et al. (2017) suggest that a template updating based on the new appearance of the object would enhance tracking performance. Therefore, using descriptors of the RoI in the latest frames provides adaptiveness about recent changes of the object.

Supported by the evidence, in the next sections we propose how to compute the *Long Term Memory* and the *Short Term Memory*. Additionally, it is presented how to combine the information from both memories into a *Combined Descriptor*.

4.4.1 COMPUTING THE LONG TERM MEMORY

The *Long Term Memory* block comprises a set of descriptors that resembles the earliest appearance of the object in the video.

The goal is finding an object representation based on reliable descriptors from several frames (using only a single frame leads to a noise representation due to the encoding of a real SNN). For this purpose, we state two assumptions:

1. The tracker provides fair outputs (BBs) for the first frames of the video.

2. The object appearance is roughly the same in these initial frames.

Therefore, considering these assumptions and recalling the property highlighted in Section 2.3.4, one possibility to obtain the *Long Term Memory* is computing the expected value E. This operation is performed of the Q first descriptors of the BBs provided by the tracker, as follows:

$$\mathbf{z}_{long}[n] = E([\mathbf{z}[0], \mathbf{z}[1], ..., \mathbf{z}[Q-1]]). \tag{4.4}$$

Notice that the *Long Term Memory* is a constant signal. Therefore, this memory block does not compute any change in the object in a frame $n \geq Q$. Figure 4.7 shows a partial view of a hypothetical *Long Term Memory* signal.

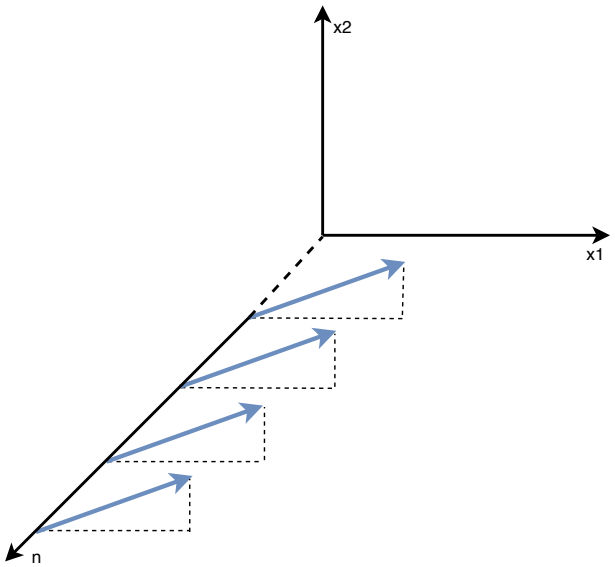


Figure 4.7: *Long Term Memory* signal representation. In this figure, it is shown only two dimensions of a single feature vector of a features map and its representation over time n.

4.4.2 COMPUTING THE SHORT TERM MEMORY

In contrast to the previous memory block, the *Short Term Memory* depends on the latest descriptors of the BBs obtained by the tracker.

This memory block is based on a filtering process. At first, there is no restriction to the type of the filter used by the *Short Term Memory* i.e., it could be any filters as adaptive, non-linear, LTI, or others. However, in this work, we restrict to the usage of LTI filters (the reason for this choice is due to evidence that this type of filter presents promising performance as it will be shown in Section 5.3.1).

As the descriptor is based on a K-dimensional feature vector, it is necessary to provide a bank of K LTI filters, as follows:

$$\mathbf{h}_{LTI}[n] = [h_{LTI_1}, h_{LTI_2}, ..., h_{LTI_K}]^T, \tag{4.5}$$

where $\mathbf{h}_{LTI}[n]$ is a LTI filter bank and each k filter is given by:

$$h_{LTI_k}[n] = \begin{cases} q_n | q_n \in \mathbb{R}, & \text{if } 0 \leq n < M, \\ 0, & \text{otherwise,} \end{cases} \tag{4.6}$$

whose properties were discussed in Section 2.1. Then, the *Short Term Memory* $\mathbf{z}_{short}[n]$

is expressed by:

$$\mathbf{z}_{short}[n] = \mathbf{z}[n] * \mathbf{h}_{LTI}[n]. \tag{4.7}$$

In contrast to $\mathbf{z}_{long}[n]$, $\mathbf{z}_{short}[n]$ changes according to the latest appearance of the tracked object, as shown in Figure 4.8.

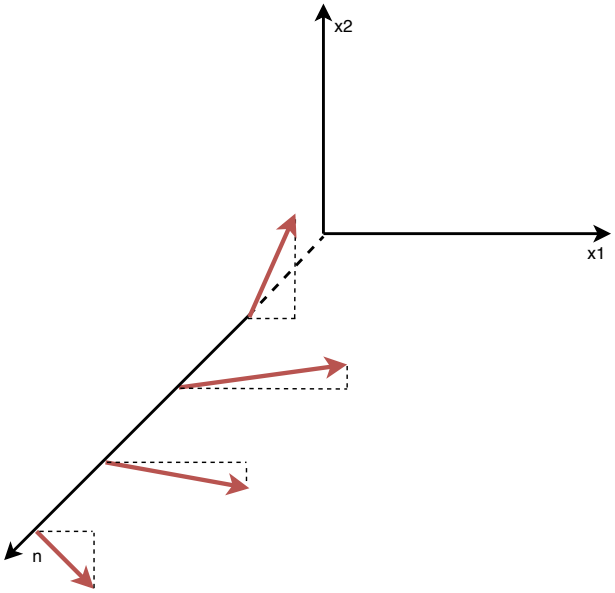


Figure 4.8: *Short Term Memory* signal representation. In this figure, it is shown only two dimensions of a single feature vector of a feature map and its representation over time n

4.4.2.1 Video Extension

The videos are extended for negative values of n according to Equation 4.8. This is done to filling the negative positions of the signal for the first $M-2$ frames when Equation 4.7 is computed. The extension is given as follows:

$$\mathbf{z}_{ext}[n] = \begin{cases} \mathbf{z}[n], & \text{if } n \geq 0; \\ \mathbf{z}[0], & \text{otherwise,} \end{cases} \tag{4.8}$$

where $\mathbf{z}_{ext}[n]$ is the signal that come from extended video. The extension provides a smooth transition from the usage of information from only a single frame (when $n = 0$) to the usage of all the last $M-1$ frames available in the video.

4.4.3 MEMORY COMBINATION

The memory blocks discussed in the previous subsections have complementary information about the RoI. Then, it is necessary to define a memory combination $\mathbb{M}$ of these blocks to generate a new descriptor, the *Combined Descriptor* $\mathbf{z}_{comb}[n]$, as defined:

$$\mathbf{z}_{comb}[n] = \mathbb{M}(\mathbf{z}_{long}[n], \mathbf{z}_{short}[n]). \tag{4.9}$$

Both memories have the same number of elements in the respective feature map and the same dimension K. Therefore, it was chosen a simple weighted sum to combine them, as indicated:

$$\mathbf{z}_{comb}[n] = \alpha\mathbf{z}_{long}[n] + (1-\alpha)\mathbf{z}_{short}[n], \tag{4.10}$$

where α is the *conservative factor* that sets the tradeoff between the reliability of the *Long Term Memory* and the adaptiveness of the *Short Term Memory*, where $\alpha \in \mathbb{R} \mid 0 \leq \alpha \leq 1$. An example of combined descriptor obtained from Equation 4.10 is shown in Figure 4.9.

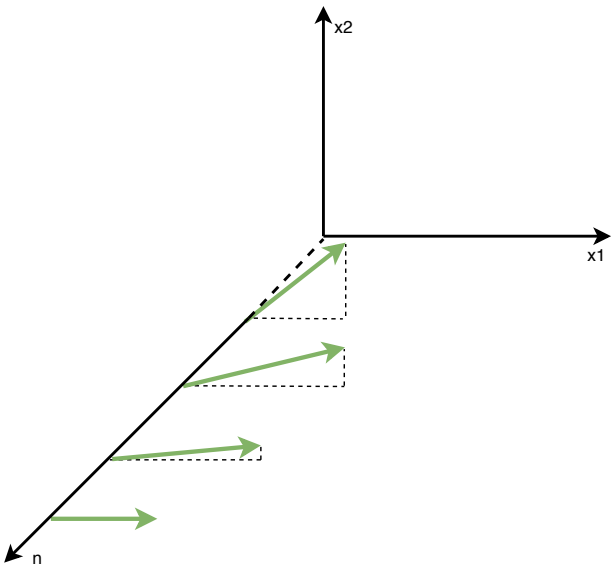


Figure 4.9: Combined Descriptor signal representation. This is the resultant combination of the *Long Term Memory* from Figure 4.7 and the *Short Term Memory* from Figure 4.8. The *conservative factor* $\alpha = 0.5$. In this figure, it is shown only two dimensions of a single feature vector of a features map and its representation over time n.

Additionally, notice that Equation 4.10 can be reduced to the original proposal of the SiamFC if $\alpha = 1$ and $Q = 1$. Alternatively, the complemented SNN is reduced to the moving average approach suggested by Valmadre et al. (2017) if $\alpha = 0$ and $\mathbf{h}_{LTI}[n]$ have

all the coefficient with the same value.

4.5 FILTER BANK ESTIMATION

In Subsections 4.4.2 and 4.4.3, it was discussed the usage of a generic LTI filter bank $\mathbf{h}_{LTI}[n]$. Now, it is shown how such filter bank is obtained through an optimization process.

Relying on the signal of descriptors $\mathbf{g}[n]$ and $\mathbf{z}[n]$ of v videos, one can find an optimized filter bank $\mathbf{h}_{LTI}[n]$. Such filter bank is used to generate the *Short Term Memory* signal $\mathbf{z}_{short}[n]$. Therefore, when $\mathbf{z}_{short}[n]$ and $\mathbf{z}_{long}[n]$ are combined into $\mathbf{z}_{comb}[n]$, the resultant descriptor must be as similar as possible to $\mathbf{g}[n]$. This can be achieved minimizing the quadratic error between these signal as shown by:

$$\mathbf{h}_{LTI}[n] = \underset{\mathbf{h}_{LTI}}{\arg\min} \left(\sum_{n,v} ||\mathbf{g}_{(v)}[n] - \mathbf{z}_{comb_{(v)}}[n]||^2 \right), \tag{4.11}$$

noticing that $\mathbf{z}_{comb_{(v)}}[n]$ is arranged from from Equations (4.7) and (4.10).

We assume that the signal of descriptors from a few videos has some components that are universally present in tracking performed by an SNN. Therefore, the goal is to use Equation (4.11) to estimate a filter bank that keeps frequency components of accurate trackings and attenuate the incorrect ones.

However, obtaining a useful filter bank is a challenging task. First, notice that BB_n varies according to $\mathbf{z}[n-1]$. Therefore, if the tracker deviates from the trajectory labeled as the correct, $\mathbf{z}[n]$ might be different from the one if it had followed the correct path. A second aspect is that the linear weighting given by the filter bank may not be efficient to recover the object descriptor from the background when the tracker fails.

The previous issues can be avoided using the SNN property discussed in Section 2.3.4. It states that similar images are encoded in the same region of feature space. Thus, restricting the training videos to those whose tracking trajectories were similar enough to the GT, it is possible to obtain a system whose input can be assumed to be independent of the previous outputs. As a result, one can also assume that a linear combination of descriptor can give a more general descriptor of the RoI.

4.5.1 FILTER LEARNING

To compute the filter bank using Equation (4.11), we have resorted to metaheuristics. In particular, we have chosen a GA to find each of the K filters of $\mathbf{h}_{LTI}[n]$ of desired filter bank.

In a first approach, one would propose a naive objective function given in Equation (4.12) to find each of the K filters $h_{kLTI}[n]$ of $\mathbf{h}_{LTI}[n]$:

$$\arg\min_{h_{kLTI}} \left(\sum_{n,v} ||g_{k(v)}[n] - z_{k_{comb}(v)}[n]||^2 \right). \tag{4.12}$$

However, since the BB_n can be critically similar to GT_n, impulsive like filters are also the solution of Equation 4.12, but they do not suit for our problem. One way to avoid the convergence for this type of undesirable solution is by adding white noise to the signal of descriptors $\mathbf{z}[n]$.

4.5.1.1 White Noise Addition

In order to avoid the improper convergence in the optimization step, it is added white noise to the signal of descriptors $\mathbf{z}[n]$, as indicated by:

$$\mathbf{z'}_{short_{(v)}}[n] = (\mathbf{z}[n] + \sigma[n]) * \mathbf{h}_{LTI}[n], \tag{4.13}$$

where $\sigma[n]$ is the white noise vector, and $\mathbf{z'}_{short_{(v)}}[n]$ is the *Corrupted Short Term Memory* signal filtered by a filter bank $\mathbf{h}_{LTI}[n]$. Thus, for the filter bank computation, Equation (4.13) replaces Equation (4.7). Finally, the *Corrupted Combined Descriptor* $\mathbf{z'}_{comb}[n]$ is given by:

$$\mathbf{z'}_{comb}[n] = \alpha \mathbf{z}_{long}[n] + (1-\alpha)\mathbf{z'}_{short_{(v)}}[n]. \tag{4.14}$$

4.5.1.2 GA Parameters

The GA is performed in each of the K filters of $\mathbf{h}_{LTI}[n]$ individually. As presented in Section 2.2, the algorithm requires: an encoding scheme for each individual; a fitness function Θ; a population P of individuals; and the genetic operators (*Selection*, *Crossover* and *Mutation*).

Each filter candidate is mapped into a *chromosome* composed of M integers, where each gene represents a filter coefficient. The fitness function Θ is the reciprical of Equation (4.12) incremented by as small number δ (it is used to guarantee a finite value of fitness) as shown as follows:

$$\Theta(h_{LTI_k}) = \sum_{v,n} \left(||g_{(v)_k}[n] - z'_{comb_{(v)_k}}[n]||^2 + \delta \right)^{-1}. \tag{4.15}$$

Once the filter encoding and the fitness function Θ are defined, the population of P individuals can be generated and the genetic operators can be performed.

First, *Selection* attempts to discard individual of lowest values of fitness and pass those who have the highest values according to the fitness function Θ in Equation (4.15).

Once the individuals are selected, the candidates mate among themselves through the *Crossover* operator. It produces a new generation of individuals, the *offspring*, whose characteristics are combined from their parents. Following, the *offspring* is submitted to the *Mutation* operator. This operation randomly changes *genes* of the *chromosome* of each individual and add *genes* that are not present in the population as a whole.

Finally, a convergence check condition is verified. If a stop condition is found, the process is halted and the filter h_{kLTI} is the one that is going to be used in our proposal. Otherwise, the genetic operators are performed again until the stop condition is verified.

The whole process previously described is performed K times to find the complete filter bank $\mathbf{h}_{LTI}[n]$. This is the one used in the complemented SiamFC.

5 EXPERIMENTAL RESULTS

This chapter evaluates the performance of the proposed method and the parameters that influence its performance. We compare our proposal to state-of-the-tracker trackers. Additionally, we present a qualitative analysis of the method.

5.1 EXPERIMENTAL SETUP

We start introducing the most important technical details used in our setup in order to implement the proposed method and verify its performance.

To run the dataset for the proposal and compute the filter bank, we used two Intel E5-4607 @ 2.20GHz processors with 24 cores and 32 GB Memory, running Ubuntu 16.04.4 LTS. The SiamFC and the proposed method were written in Python3.5. As additional libraries, we used: TensorFlow 1.10.0, for the implementation of the SiamFC, Numpy 1.14.5, for numerical computation, OpenCV 3.4.3, data visualization and the evolutionary computation framework DEAP 1.2.2, for the computation of the GA.

The external sources for the version of the SiamFC used in this work is available at: https://github.com/www0wwwjs1/tensorflow-siamese-fc, the Online Tracking benchmark (OTB) dataset can be accessed at: http://cvlab.hanyang.ac.kr/tracker_benchmark/ and the VOT2015 dataset at: http://www.votchallenge.net/vot2015/dataset.html. We perform the hypothesis test using *scikit-posthocs* provided by Terpilowski (2019).

5.1.1 DATASETS

It was used two datasets in this work. The first one, the VOT2015, was used to design the filters of our proposal. The second one, the OTB, was used to evaluate the performance of the proposed method and compare it to state-of-the-art trackers.

The VOT2015 is a labeled dataset composed of 60 uncategorized videos. It presents challenging videos that include: occlusion; background clutter; illumination change; object deformation and rotation. There is no uniformity for frame rate and resolution, and the dataset is composed only of short term videos, which are ideal for the filter bank design, as it would demand less computational resources to find the optimized filters.

The second dataset is OTB. It is composed of 50 categorized videos according to 11 attributes: Abrupt Motion; Background Clutter; Blur; Deformation; Illumination Variation; In-Plane Rotation; Low Resolution; Occlusion; Out-of-Plane Rotation; Out-of-View; Scale Variation. One video can belong to more than one category. There is also no uniformity for frame rate and resolution, and there are short and long term videos. As OTB is one of the most recognized datasets in the literature (BEI et al., 2018), we use it as the evaluation benchmark for our work.

5.1.1.1 Evaluation protocol

The OTB dataset provides a specific evaluation protocol. For a given RoI in the first frame, the tracker under evaluation should output the following BBs. Each of these has to corresponds to the RoIs in the following frames. The error between the GT and the tracker's output is computed over the video. Two metrics evaluate tracker performance: Location Error (LE) and Intersection-over-Union (IoU). Figure 5.1 shows a graphic example of both metrics, and they are explained as follows:

1. LE: This metric takes into account the distance between the BB outputted by the tracker and the GT. Its computation is given by the Euclidean distance given by:

$$LE = ||BB_c - GT_c||, \tag{5.1}$$

   where BB_c and GT_c is the center of the BB and the center of the GT, respectively. The closest the BBs of the analyzed frame are to the GT, the lower the LE will be.

2. IoU: This metric takes into account the overlap between the BB and the GT. It is defined as follows:

$$IoU = \frac{BB \cap GT}{BB \cup GT}, \tag{5.2}$$

   where $\cap$ and $\cup$ represent the intersection and the union, respectively, of the BB and GT region. The better the overlap between the GT and the outputted BB for a given frame, the closer it is to 1.

In order to analyze the performance for a given tracker in a set of videos based on the two metrics presented above, one can estimate two different types of curves:

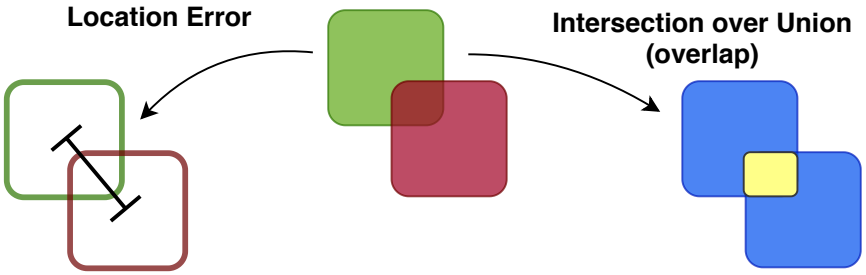


Figure 5.1: Representation of LE and IoU.

1. Precision plot: This curve shows the proportion of frames successfully tracked for a given LE threshold. Then, it is a graph of precision vs. location error threshold.

2. Success plot: This curve shows the proportion of frames that achieved a minimum value of IoU for a given threshold. Then, it is a graph of success rate vs. overlap threshold.

A good way to generalize the evaluation of these curves is to compute the Area under the curve (AUC). The greater the area, the better is the performance for this metric.

The AUC is widely used to compare the performance of different trackers. However, for a complete evaluation, the analysis of the plots is also recommended.

5.2 PARAMETER SETTING

This section presents values for the parameters used in this proposal. We also present those which achieve the best performance in our evaluation. Additionally, we discuss the impact of the main parameters in the global performance of this work.

The GA specification for the optimization of the filter bank of order M is: the probability of gene mutation is defined to be $p_g = \frac{1}{M}$; the probability of selecting an individual for mutation is $p_m = 90\%$; the mate operation is the two-point crossover, which probability is $p_c = 95\%$; the population size P is $5M$; the convergence criteria is set to be the maximum number of iterations M_i allowed for convergence, where $M_i = 600$ and finally we adopted the integer encoding for the filter coefficients;.

Ideally, the number of descriptors Q of the *Long Term Memory*; the order M of the filter bank that is used by the *Short Term Memory*; the *Conservative Factor* α and the white noise standard deviation σ would be better determined through *grid search*. However, due to the high computational cost of the learning step, we empirically analyzed

the impact of each parameter. Thus, we fixed the parameters which presented the best performance to the method, and we changed each one at the time to verify its impact on the proposal. Therefore, the parameters were fixed, unless otherwise specified, as: $Q = 30$, $M = 31$, $\alpha = 0.65$, $\sigma = 0.9$. Additionally, the default training videos from VOT2015 were: "bag", "racing", "ball1", "octopus", "bolt2", "pedestrian2", "road".

5.2.1 LONG TERM MEMORY EVALUATION

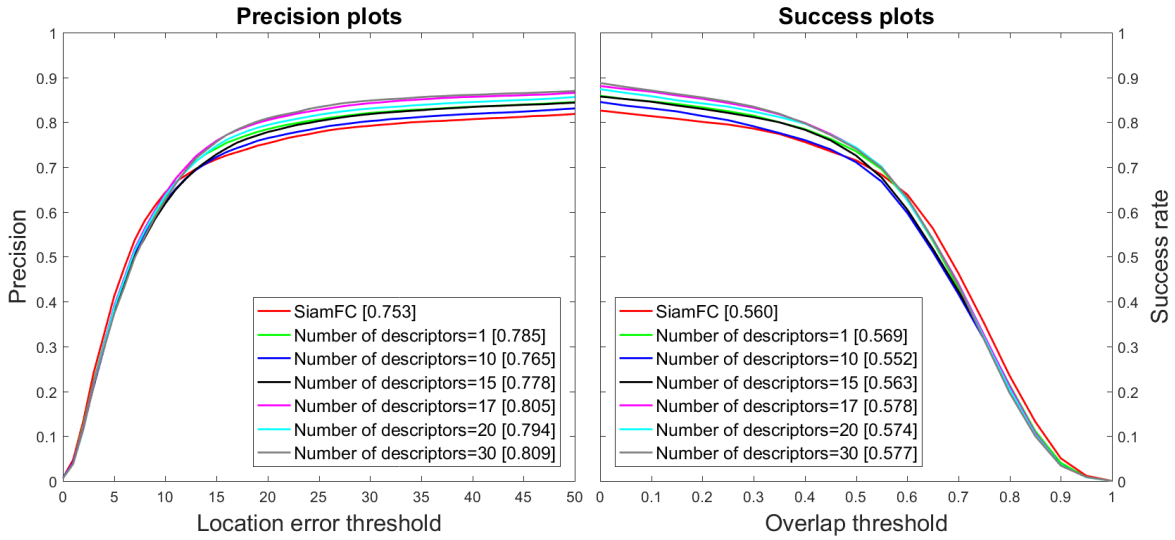


Figure 5.2: Analysis of the number of descriptor that composes the *Long Term Memory*. The respective AUCs are shown in the legend between brackets.

The number of descriptors Q that compose the *Long Term Memory* was analyzed. According to Figure 5.2, the most appropriate size for this memory according to the AUC of the precision and the success plots was $Q = 30$. However, for most values of Q, the proposed method has higher AUCs than the SiamFC.

5.2.2 SHORT TERM MEMORY EVALUATION

The second parameter analyzed was the order M of the filter bank. This parameter defines the number of descriptors used by the *Short Term Memory* to output its evaluation. Notice in Figure 5.3 that all the values of M outperforms the proposal the SiamFC. However, in these scenarios, $M = 31$ is undoubtedly the most appropriate value for our proposal.

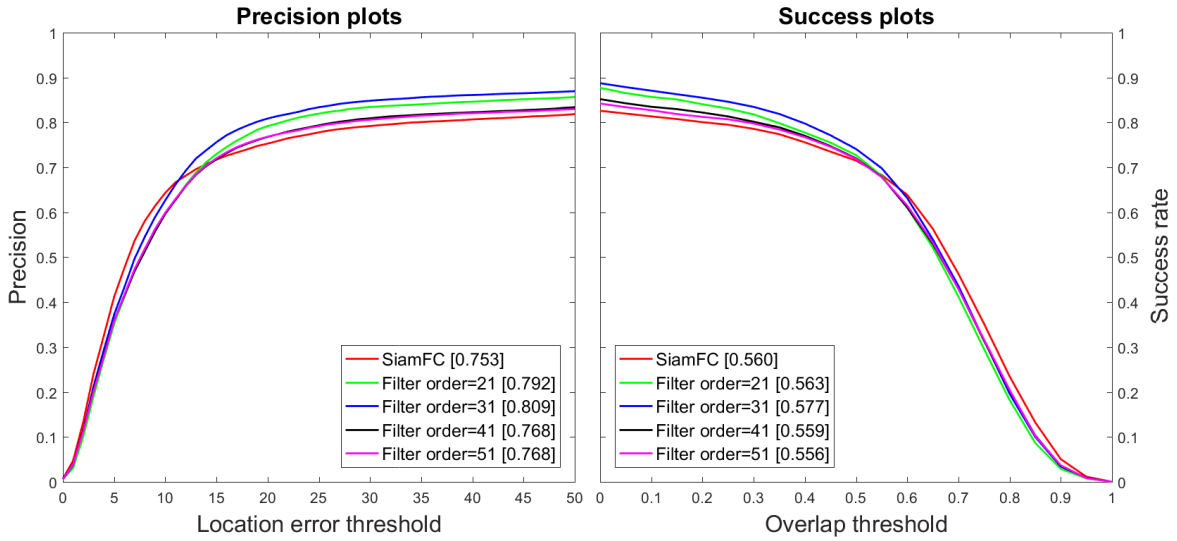


Figure 5.3: Analysis of the filter order of the filter bank used by *Short Term Memory*. The respective AUCs are shown in the legend between brackets.

5.2.3 WHITE NOISE EVALUATION

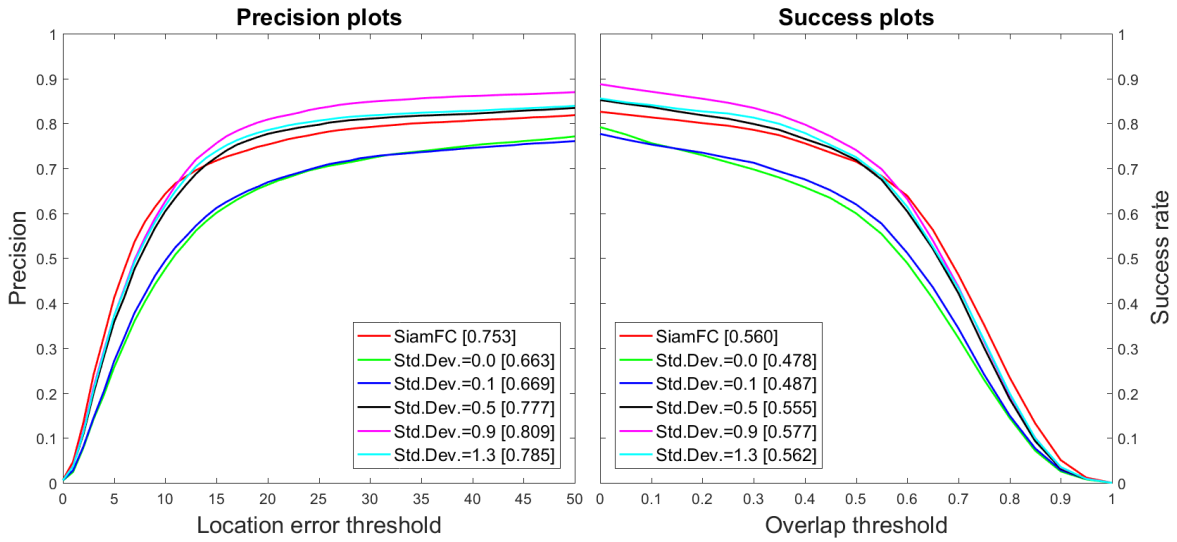


Figure 5.4: Impact of the noise standard deviation in the computation of the trained filters compared to the baseline. The respective AUCs are shown in the legend between brackets.

In contrast to size of the *Long Term Memory* and the *Short Term Memory*, the standard deviation σ of the white noise added to $\mathbf{z}[n]$ is a sensitive parameter. In Figure 5.4, observe that adding no noise, $\sigma = 0$, or even adding noise with low standard deviation, as $\sigma = 0.1$, results in poor performance of the proposed method. These parameters value have inferior performance when compared to the baseline. Increasing the value of σ, the

performance also increases, up to the value $\sigma = 0.9$. Beyond these value, the system's performance decreases, as the noise severely corrupts the signal.

5.2.4 CONSERVATIVE FACTOR EVALUATION

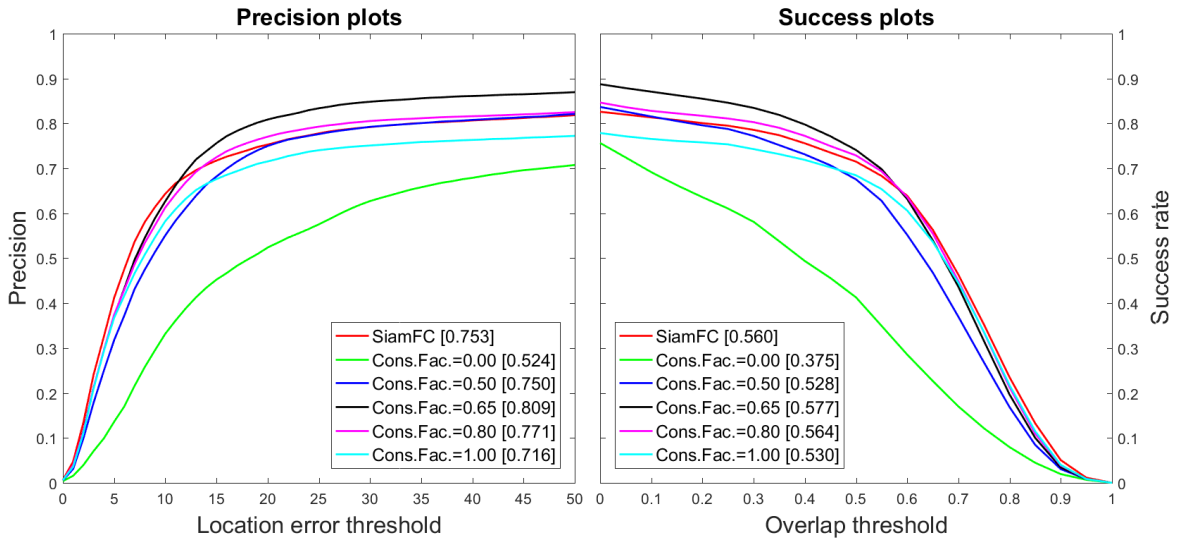


Figure 5.5: Impact of the conservative factor in the computation of the filter compared to the baseline. The respective AUCs are shown in the legend between brackets.

An even more sensitive parameter is the *conservative factor* α. Notice in Figure 5.5 that if the *Combined Descriptor* relies only in the *Short Term Memory* ($\alpha = 0$) or in the *Long Term Memory* ($\alpha = 1$), our proposal cannot reach the baseline performance. However, when α is set to 0.80 or 0.65, the proposed method can outperform the SiamFC, where $\alpha = 0.65$ presents the highest performance according tho the AUCs of precision and success plots.

5.2.5 SELECTION OF VIDEOS FOR TRAINING

We chose three scenarios, where we selected the videos used for training the filter bank:

1) Videos whose tracker output BB_n has high intersection to GT_n.

2) Videos whose tracker output BB_n has low intersection to GT_n.

3) Videos whose tracker output BB_n has high intersection to GT_n and videos whose BB_n has low intersection to GT_n.

The metric elected to indicate the intersection between GT_n and BB_n was the F-Measure. It is the harmonic mean between the *precision*[1] p and *recall*[2] r of a video, as expressed in the following expression:

$$F\text{-}Measure = \frac{2 \cdot p \cdot r}{p + r}. \tag{5.3}$$

The higher the F-Measure F, the fairer the tracker followed the RoI. Therefore, if BB_n intersects GT_n with a IoU greater than a threshold Γ for most part of the trajectory, the F-Measure F is close to 1.

Thus, in order to validate these scenarios, we selected videos in the VOT2015 dataset that meet the following criteria: *scenario 1*, the tracker output BB_n for the videos have a F-measure $F \geq 0.80$; *scenario 2*, BB_n for the selected videos have $F \leq 0.25$; *scenario 3*, BB_n of the selected video comprises $F \geq 0.80$ and $F \leq 0.25$. In all scenarios, we set the IoU threshold $\Gamma = 0.25$. Table 5.1 shows the selected videos that meet these requirements.

Scenario	**Videos**
1	''bag'',''racing'',''ball1'', ''octopus'', ''bolt2 '',''pedestrian2'',''road''
2	''bolt1'',''fish2'', ''handball1'', ''leaves'',''nature'',''rabbit'',''singer2''
3	''bag'',''racing'',''ball1'', ''octopus'', ''bolt1'',''fish2'',''handball1''

Table 5.1: Videos in the VOT2015 dataset that meet the criteria for scenario 1, 2 and 3.

Figure 5.6 shows the performance when the filter bank is trained in the scenarios. *Scenario 1* presented the best result among the three. In *scenario 2*, the filters also showed a satisfactory performance when evaluated, although it does not outperform *scenario 1*. *Scenario 3* showed a less expressive performance when compared to the previous ones. The results might indicate that training the filters with BB_n with high F-Measure to GT_n can be positive in the computation process. However, the relatively high performance of the *scenario 2* inquires if BB_n not tightly close to the GT_n can also contribute to the computation of the filter bank. All in all, the impact of the tracking quality over the videos is still one open-end question and requires a more detailed analysis.

[1]Number of BBs outputted by the tracker whose IoU to the GT is greater then a given threshold Γ divided by the total amount of BB detected in a video.

[2]Number of BBs outputted by the tracker whose IoU to the GT is greater then a given threshold Γ divided by the total number of BB that the GT presented as a valid.

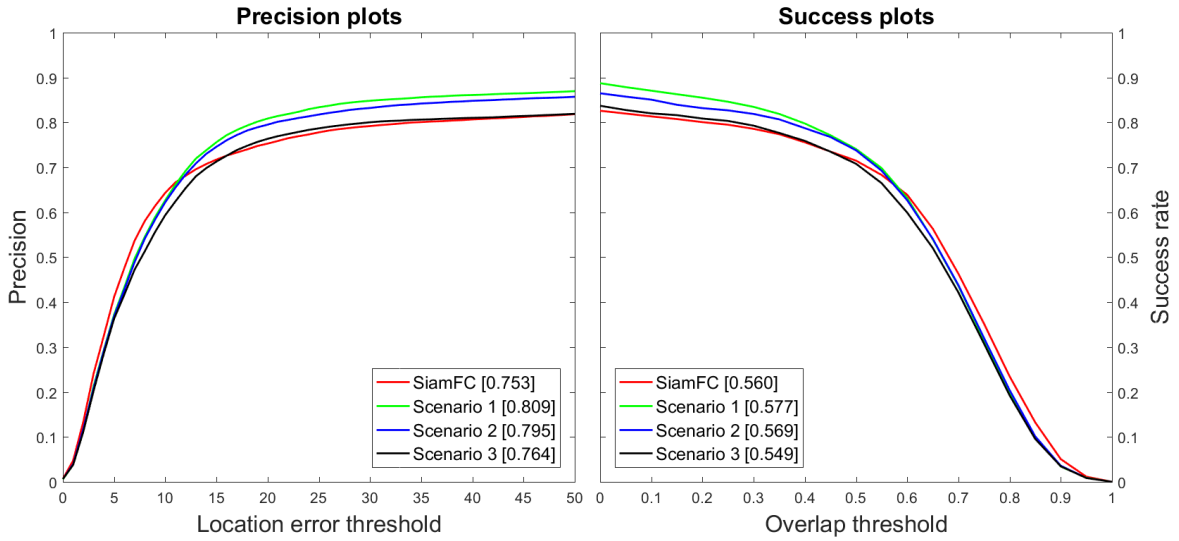


Figure 5.6: Performance of the filter bank trained with videos tracked with high F-Measure values. The respective AUCs are shown in the legend between brackets.

5.3 METHOD EVALUATION

After finding the most appropriate parameter values for the proposed method, we evaluate the performance of standard LTI filters, and we finally proceed to a more in-depth comparison to the baseline.

5.3.1 COMPARISON TO STANDARD FILTERS

In this section, we analyze our proposal using two standard filters for the filter bank of the *Short Term Memory*: the Moving Average and the Gaussian filters.

The first filter analyzed is the Moving Average mentioned by Valmadre et al. (2017). We verified that it does not show significant improvements when compared to the original framework of the SiamFC. We observed that for some particular videos, the moving average presented an improvement of performance compared to the original SiamFC proposal. However, its overall performance on the OTB dataset, depicted in Figure 5.7, shows that this approach does not present gains compared to the SiamFC. This result led us to inquire if the filter could be composed of different coefficients. Then, we figured out that a Gaussian filter presents a better performance when compared to the standard Moving Average, as also shown in Figure 5.7. The Gaussian shape filter is given as follows:

$$h[n] = \frac{e^{-\frac{(n-\lfloor M/2 \rfloor+1))^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}, \tag{5.4}$$

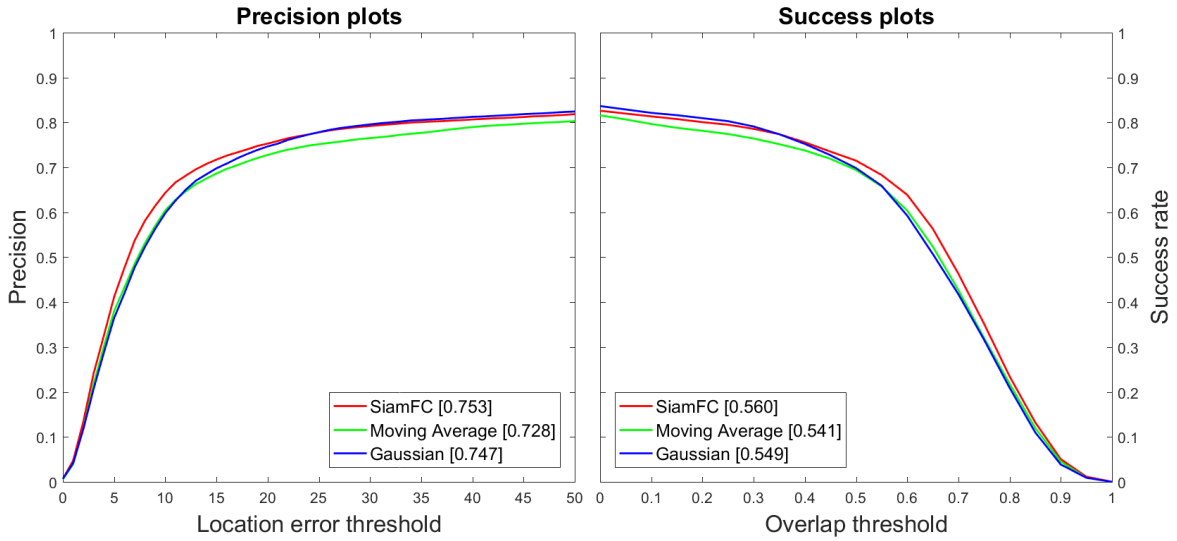


Figure 5.7: Precision and success plots for Moving Average and the Gaussian Filters compared to the SiamFC. The respective AUCs are shown in the legend between brackets.

where σ defines the width of the Gaussian distribution shape, and M is the order of the filter.

5.3.2 BASELINE COMPARISON

In this subsection, we deploy a comparison between the proposed method to the baseline, the SiamFC.

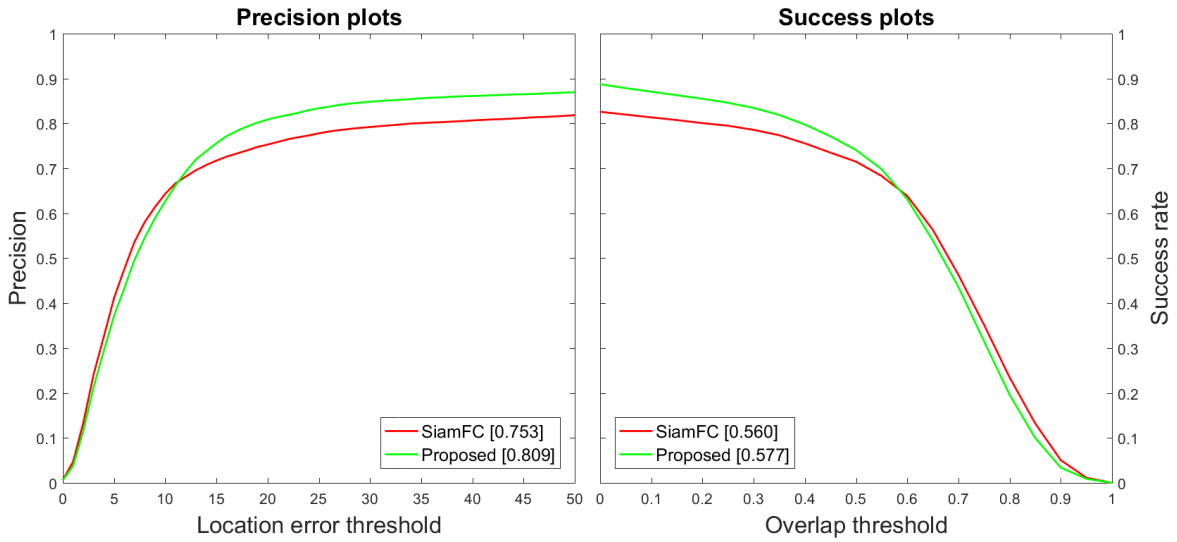


Figure 5.8: Precision and success plots for the proposed method an the baseline. The respective AUCs are shown in the legend between brackets.

The precision and success plots for both methods are shown in Figure 5.8. Our method

outperforms the baseline according to the AUC of the curves, setting it as a default choice for a general application. Notice that the AUC improvement is about 7.4% and 3.0% for precision and success plots, respectively. Our proposal also has a substantial gain of performance from moderate to high LE threshold tolerance and low values of IoU threshold. The baseline has a slightly better performance for low LE threshold and high IoU threshold. The mean of our proposal is also superior to the baseline for both curves. These metrics and the standard deviation for the curves can be verified in Table 5.2 and Table 5.3.

Precision Plot / **Method**	AUC	Mean	Std. Dev.
SiamFC	0.753	0.688	**0.200**
Proposed	**0.809**	**0.721**	0.227

Table 5.2: Summary of the precision plots performance according to AUC and the statistical information of mean and standard deviation. It is highlighted in bold the best performance for each metric.

Success Plot / **Method**	AUC	Mean	Std. Dev.
SiamFC	0.560	0.559	**0.293**
Proposed	**0.577**	**0.577**	0.321

Table 5.3: Summary of the success plots performance according to AUC and the statistical information of mean and standard deviation. It is highlighted in bold the best performance for each category for each of the thresholds.

As the F-Measure comprises information about false positives and false negatives rate of a tracker, we compared both methods according to this metric, as shown in Table 5.4. Notice that for a loose threshold ($\Gamma = 0.25$), the proposed method outperforms the baseline, showing a better tracker *adherence* to the RoI, i.e., it fails less frequently. In contrast, the original proposal of the SiamFC locates the RoI more precisely when it does not fail, as it has a greater F-Measure for a higher Threshold ($\Gamma = 0.75$).

For an improved analysis, we also performed the *post-hoc* Nemenyi test to the curves of SiamFC and our method. The precision rate of location has p-value $= 4.95 \cdot 10^{-4}$, meaning that both curves are statistically different. However, the p-value for the success rate of overlap is 0.406 meaning that the curves are statistically similar.

Finally, we analyzed the performance of our proposal in the 11 categories of the OTB

Method \ Threshold	0.25	0.50	0.75
SiamFC	0.612	0.460	**0.188**
Proposed	**0.642**	**0.462**	0.133

Table 5.4: F-measure for different thresholds. It is highlighted in bold the best performance for each threshold.

according to LE precision and the IoU success. Table 5.5 and Table 5.6 show the precision and the success, respectively, for different thresholds.

	Threshold	50		30		10	
	Method	PM	BL	PM	BL	PM	BL
Category	Abrupt Motion	**0.877**	0.830	**0.837**	0.795	0.469	**0.580**
	Background Clutter	**0.835**	0.763	**0.817**	0.727	**0.645**	0.575
	Blur	**0.846**	0.816	**0.806**	0.787	0.420	**0.551**
	Deformation	**0.847**	0.784	**0.831**	0.763	0.556	**0.568**
	Illumination Variation	**0.855**	0.769	**0.829**	0.741	0.534	**0.571**
	In-Plane Rotation	**0.836**	0.772	**0.804**	0.739	0.570	**0.592**
	Low Resolution	**0.819**	0.782	**0.776**	0.744	0.531	**0.548**
	Occlusion	**0.842**	0.795	**0.819**	0.775	0.530	**0.592**
	Out-of-Plane Rotation	**0.848**	0.792	**0.824**	0.763	0.590	**0.608**
	Out-of-View	**0.735**	0.713	0.689	**0.693**	0.358	**0.481**
	Scale Variation	**0.883**	0.834	**0.863**	0.804	0.670	**0.682**

Table 5.5: LE precision for different thresholds. The first row corresponds to thresholds values. The second row indicates the method, the proposed method (PM) or the baseline (BL). From the third row, each line corresponds to the performance of one category of the OTB dataset. It is highlighted in bold the best performance for each category for each of the thresholds.

The data in Table 5.5 reinforces that our method is more reliable to indicate the RoI center than the baseline, presenting better *adherence*. In contrast, for an application where precision is required, the baseline presents better performance. This observation is valid for all the 11 categories analyzed.

Alternatively, Table 5.6 shows that our proposal outperforms the baseline for all categories of low values of threshold (about 0.25), and it is competitive for medium values of threshold (0.50). In contrast, the baseline outperforms the proposal for high values of thresholds (about 0.75). It shows that our method intersects to the RoI more frequently than the baseline, showing again that the proposed method has a better *adherence* to the

	Threshold	0.25		0.50		0.75	
	Method	PM	BL	PM	BL	PM	BL
Category	Abrupt Motion	**0.857**	0.817	**0.692**	0.686	0.260	**0.307**
	Background Clutter	**0.809**	0.727	**0.736**	0.636	**0.388**	0.321
	Blur	**0.812**	0.794	0.623	**0.666**	0.207	**0.291**
	Deformation	**0.834**	0.764	**0.684**	0.648	0.239	**0.258**
	Illumination Variation	**0.814**	0.739	**0.668**	0.649	0.291	**0.341**
	In-Plane Rotation	**0.804**	0.751	**0.697**	0.675	0.302	**0.336**
	Low Resolution	**0.795**	0.762	0.666	**0.674**	0.257	**0.317**
	Occlusion	**0.816**	0.774	**0.687**	0.678	0.238	**0.297**
	Out-of-Plane Rotation	**0.819**	0.765	**0.699**	0.672	0.290	**0.318**
	Out-of-View	**0.730**	0.708	0.638	**0.645**	0.312	**0.387**
	Scale Variation	**0.856**	0.813	0.716	**0.721**	0.308	**0.368**

Table 5.6: IoU success for different thresholds. First row corresponds to thresholds values. Second row indicates the method, the proposed method (PM) or the baseline (BL). From the third row, each line corresponds to the performance of one category of the OTB dataset. It is highlighted in bold the best performance for each category for each of the thresholds.

RoI. However, when the SiamFC successfully intersects the RoI, it does more precisely.

5.3.3 COMPARISON WITH THE STATE-OF-THE-ART

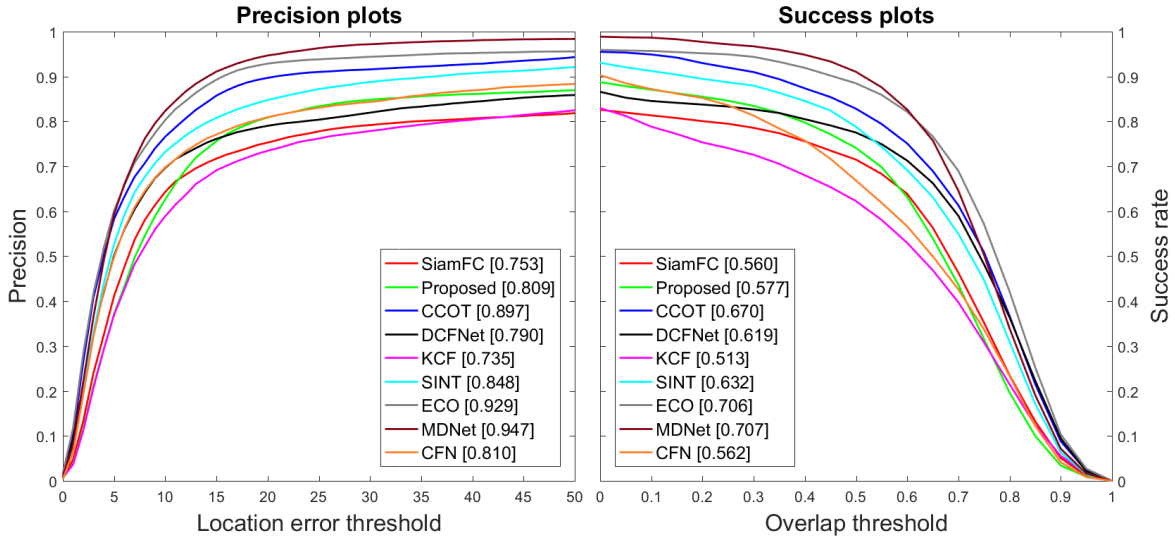


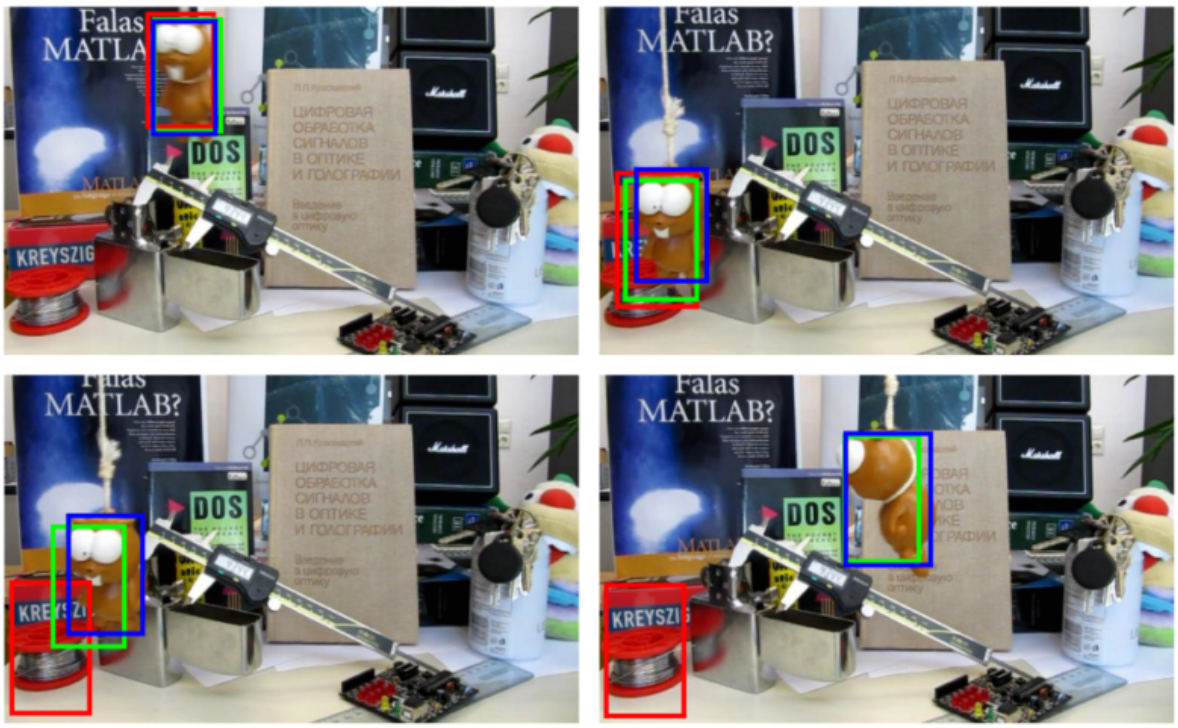
Figure 5.9: Precision and success curves for the performance of the proposed method (green), compared to state-of-the-art methods. The respective AUCs are shown in the legend between brackets.

The performance of several state-of-the-art trackers discussed in Chapter 3, whose data

performance on OTB dataset are available online, is shown in Figure 5.9. The proposed method was able to improve the performance of the SiamFC, making it competitive to the trackers: DCFNet and the CFN. Our proposal outperforms these trackers in one of the curves according to the AUC. In contrast, notice that MDNET, CCOT, ECO, and SINT have higher AUC than our tracker. Due to its simple and effective approach, however, our method is lightweight with fairly comparable results. We remark that the lack of object occlusion detection greatly impairs our precision and success rates.

5.4 VISUAL PERFORMANCE

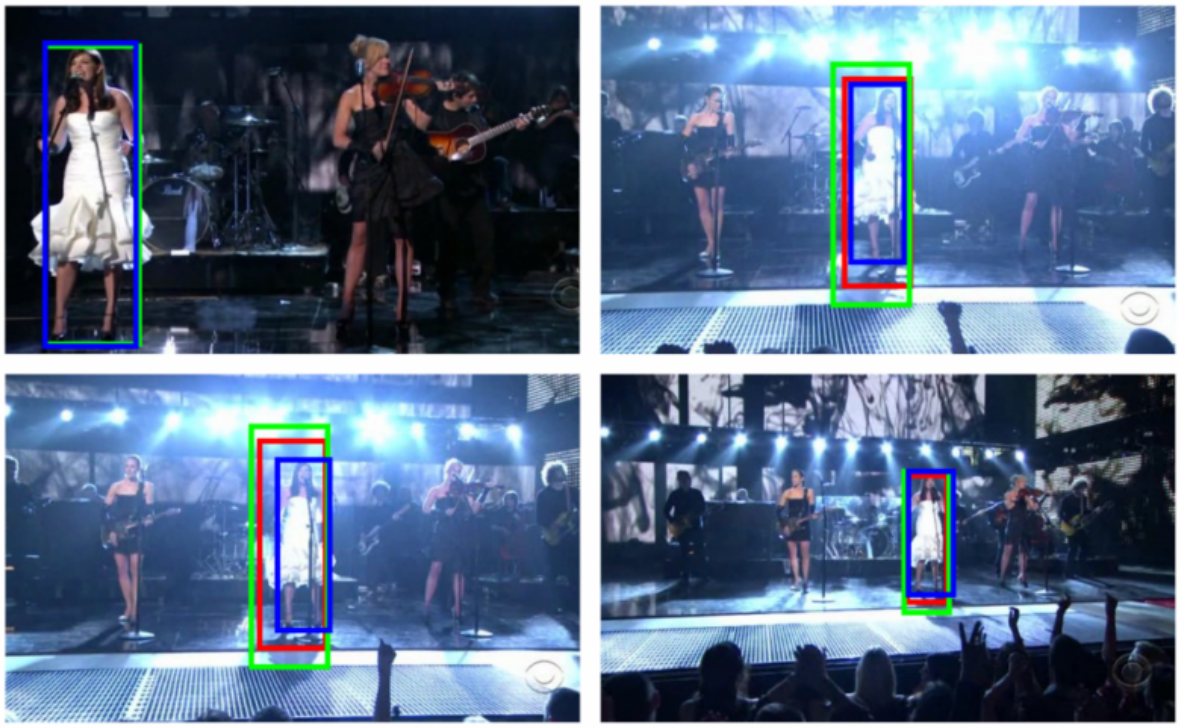
We present a few sequences that let the reader interpret the results presented in Subsection 5.3.2. Figure 5.10a and Figure 5.10b show frames that the proposed method outperforms the baseline. SiamFC confuses the background to the foreground and it is not able to recover from failure anymore. Notice that although the proposed method does not always show a precise location of the RoI in the images, it points out, even in a coarsely way, the RoI (good *adherence*), while the SiamFC fails.



(a)

Alternatively, Figure 5.10c helps us to understand why the SiamFC show better performance for high values of threshold in the success plots and low ones in the precision

(b)



(c)

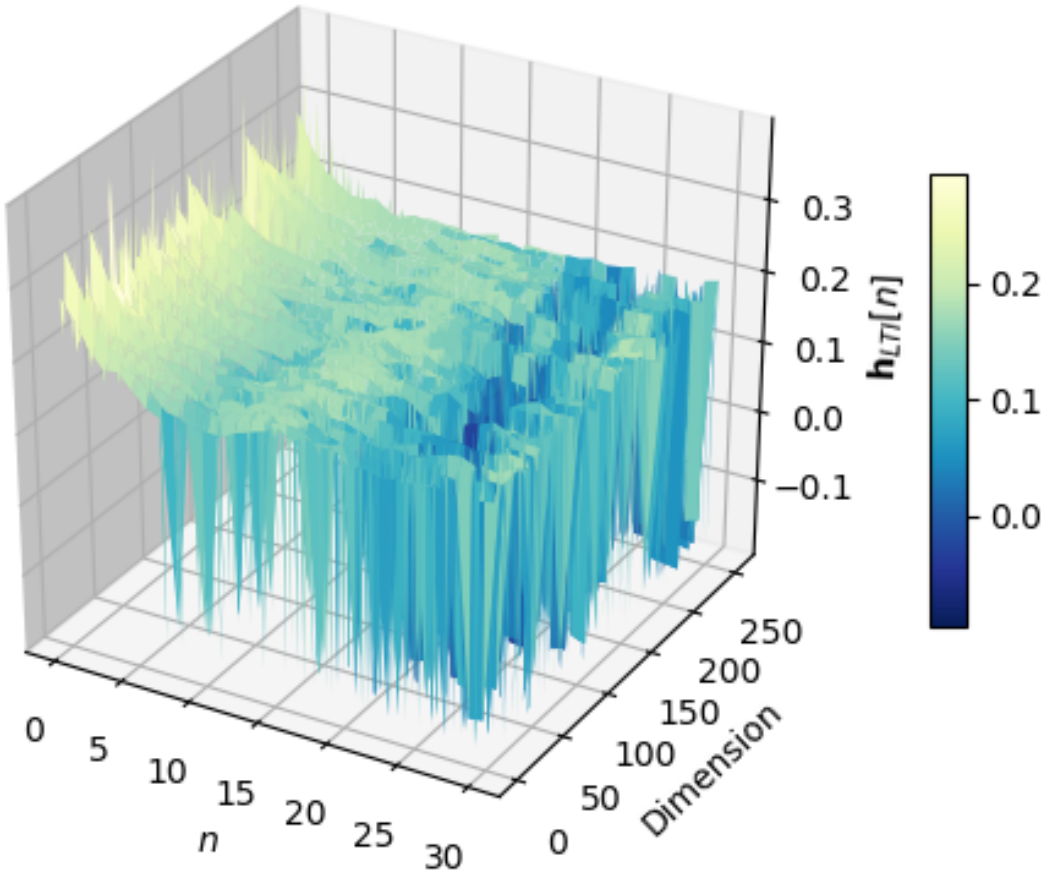
Figure 5.10: Tracking performance comparison between: GT (in blue), SiamFC (in red) and the proposed method (in green). All the sequences are from the OTB dataset: (a) ''lemming1''(b) ''skiing''(c) ''singer1''.

plots. Both SiamFC and the proposed method successfully indicate the RoI, however the SiamFC does it better due to its fair precision.

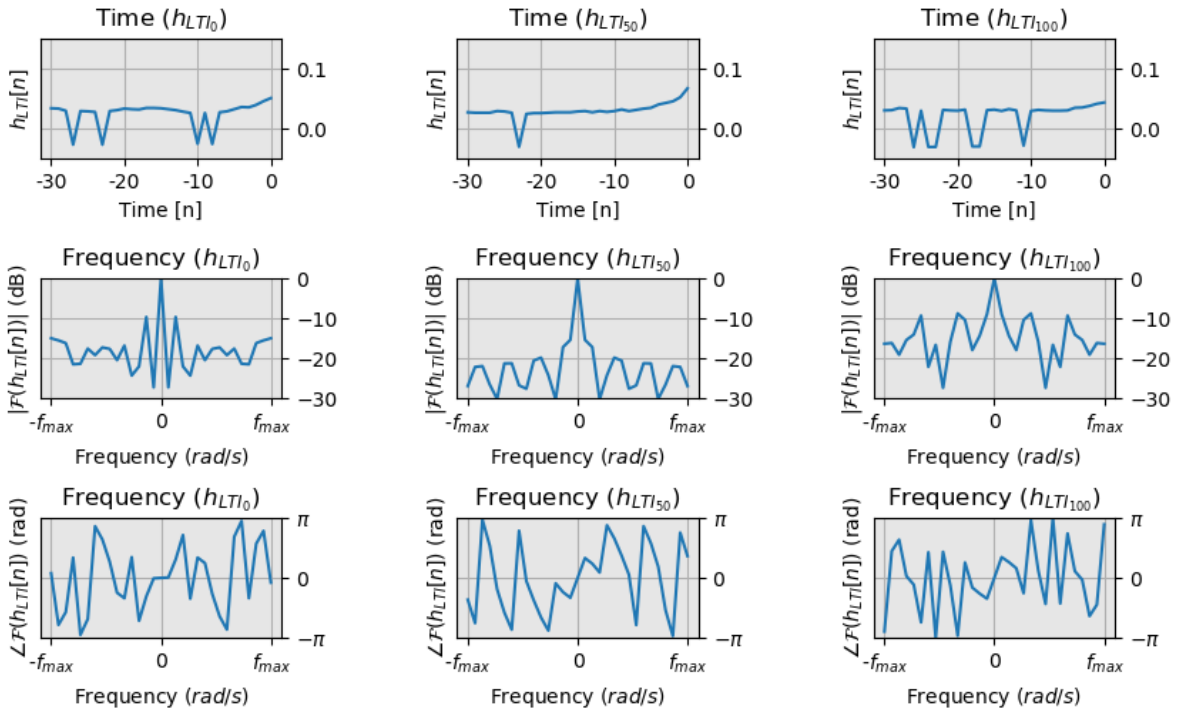
5.5 LEARNED FILTER ANALYSIS

We also discuss the results of the trained filter bank from a qualitative view. Figure 5.11a shows an overview of the filter bank. Notice that each filter has a strong DC component, and most of the filters have the first coefficient as the greater one.

Figure 5.11b shows three filters from the bank to a more in-depth analysis. Notice the strong DC component in all the three filters that can be verified by applying the Fast Fourier Transform (FFT). Although we have discussed the limits of performance of an SNN, one of its fundamental characteristics is keeping the descriptor insensitive for different instances of the objects. Therefore, we expected beforehand that this strong DC component would be present in the filter bank.



(a)



(b)

Figure 5.11: Filter bank analysis (a) Shape of all the 256 filters of the bank. (b) Filters corresponding to dimensions $k = 0, 50, 100$ of the bank shown individually for time and frequency analysis.

Another important aspect are the strongest coefficients of the filters. They are usually the first ones. This characteristic shows that the last descriptors in the video are the most relevant ones. It is an intuitive consideration, as the change in the appearance of RoI in the latest frames looks to be more important than the previous appearance of the RoI.

A final consideration that only can be observed when analyzing each filter is that for most of them, the low-frequency components tend to be higher than the high-frequency ones. It can be understood as a fast change in RoI appearance is less important than a consistent and smoother one.

6 CONCLUSION

In this work, an SNN was complemented to improve tracking based on the concept of memory blocks. We proposed two of them: the *Long Term Memory* and the *Short Term Memory*. The first block is composed of initial descriptors obtained by the SNN, while latest descriptors compose the second block. The blocks provide an adaptive nature to the tracker, yielding in an improved performance to long term tracking. Although we have implemented this proposal in the SiamFC framework, it is possible to use it on any SNN. We also showed how to compute the memory blocks output, and we focused on the computation of a filter bank, based on GA, for the *Short Term Memory*.

Compared to the baseline, our proposal has better global performance, according to AUC of the OTB dataset. Additionally, our method has greater adherence to the objects tracked over time. The proposal also achieves comparable performance to the state-of-the-art trackers, presenting a high performance at the same it requires a low computational cost.

Although the proposed method enhances the SiamFC, it is not able to handle RoI occlusion, which impairs tracking capability. Additionally, as the *Short Term Memory* uses an LTI filter bank, this block may not capture nonlinear information of the RoI. Finally, SiamFC also presents higher performance on low tolerance location of the object.

6.1 FUTURE WORK

Direction for future works suggests more sophisticated approaches for exploration of temporal series, including Long Short-Term Memory (LSTM) NN, DNN, and Markov Chain. Additionally, the use of more memory blocks and alternative ways to combine them seems promising. Due to the high computational cost to compute the filters, one can explore a systematic way to combine parameters in the optimization problem. Moreover, the random search might present good results to find the best parameters to the method. It would replace manual search, as done in this work, or grid search, as traditionally done in ML literature (BERGSTRA; BENGIO, 2012).

Additionally, occlusion detection strategies can be used to avoid erroneous learning of recent object appearance. Also, a better understanding of the videos selected for

tracking may show alternatives to improve general performance. Finally, another point that deserves efforts is the exploration of other optimization algorithms to obtain the filter bank.

REFERENCES

AHMAD, S. U.; ANTONIOU, A. Cascade-form multiplierless fir filter design using orthogonal genetic algorithm. In: IEEE. **2006 IEEE International Symposium on Signal Processing and Information Technology**, 2006. p. 932–937.

BALDI, P.; CHAUVIN, Y. Neural networks for fingerprint recognition. **Neural Computation**, v. 5, 05 1993.

BEI, S.; ZHEN, Z.; WUSHENG, L.; LIEBO, D.; QIN, L. Visual object tracking challenges revisited: Vot vs. otb. **PloS one**, Public Library of Science, v. 13, n. 9, p. e0203188, 2018.

BERGSTRA, J.; BENGIO, Y. Random search for hyper-parameter optimization. **Journal of Machine Learning Research**, v. 13, n. Feb, p. 281–305, 2012.

BERTINETTO, L.; VALMADRE, J.; HENRIQUES, J. F.; VEDALDI, A.; TORR, P. H. Fully-convolutional siamese networks for object tracking. In: SPRINGER. **European conference on computer vision**, 2016. p. 850–865.

BROMLEY, J.; GUYON, I.; LECUN, Y.; SÄCKINGER, E.; SHAH, R. Signature verification using a" siamese" time delay neural network. In: **Advances in neural information processing systems**, 1994. p. 737–744.

CARREIRA, J.; ZISSERMAN, A. Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. In: IEEE. **IEEE Conference on Computer Vision and Pattern Recognition**, 2017. p. 4724–4733.

CEMES, R.; AIT-BOUDAOUD, D. Genetic approach to design of multiplierless fir filters. **Electronics Letters**, IET, v. 29, n. 24, p. 2090–2091, 1993.

CHOI, J.; CHANG, H. J.; YUN, S.; FISCHER, T.; DEMIRIS, Y.; CHOI, J. Y. Attentional correlation filter network for adaptive visual tracking. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**, 2017. p. 4807–4816.

CUN, Y. L.; MATAN, O.; BOSER, B.; DENKER, J. S.; HENDERSON, D.; HOWARD, R. E.; HUBBARD, W.; JACKET, L.; BAIRD, H. S. Handwritten zip code recognition with multilayer networks. In: IEEE. [**1990**] **Proceedings. 10th International Conference on Pattern Recognition**, 1994. v. 2, p. 35–40.

DANELLJAN, M.; BHAT, G.; KHAN, F. S.; FELSBERG, M. et al. Eco: Efficient convolution operators for tracking. In: **CVPR**, 2017. v. 1, n. 2, p. 3.

DANELLJAN, M.; ROBINSON, A.; KHAN, F. S.; FELSBERG, M. Beyond correlation filters: Learning continuous convolution operators for visual tracking. In: SPRINGER. **European Conference on Computer Vision**, 2016. p. 472–488.

DEY, A. K.; SAHA, S.; SAHA, A.; GHOSH, S. A method of genetic algorithm (ga) for fir filter construction: design and development with newer approaches in neural network platform. **International Journal of Advanced Computer Science and Applications**, Citeseer, v. 1, n. 6, p. 87–90, 2010.

GUO, Q.; FENG, W.; ZHOU, C.; HUANG, R.; WAN, L.; WANG, S. Learning dynamic siamese network for visual object tracking. In: **The IEEE International Conference on Computer Vision (ICCV).(Oct 2017)**, 2017.

HAUPT, R. L.; HAUPT, S. E. Practical genetic algorithms. Wiley Online Library, 2004.

HAYKIN, S. **Neural networks: a comprehensive foundation**, 1994.

HELD, D.; THRUN, S.; SAVARESE, S. Learning to track at 100 fps with deep regression networks. In: SPRINGER. **European Conference on Computer Vision**, 2016. p. 749–765.

HENRIQUES, J. F.; CASEIRO, R.; MARTINS, P.; BATISTA, J. High-speed tracking with kernelized correlation filters. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 37, n. 3, p. 583–596, 2014.

HOLLAND, J. Adaptation in natural and artificial systems: an introductory analysis with application to biology. **Control and artificial intelligence**, University of Michigan Press, 1975.

HUANG, Z. An investigation of deep tracking methods. In: IEEE. **Technologies and Applications of Artificial Intelligence (TAAI), 2017 Conference on**, 2017. p. 58–61.

KALAL, Z.; MIKOLAJCZYK, K.; MATAS, J. et al. Tracking-learning-detection. **IEEE transactions on pattern analysis and machine intelligence**, v. 34, n. 7, p. 1409, 2012.

KOCH, G.; ZEMEL, R.; SALAKHUTDINOV, R. Siamese neural networks for one-shot image recognition. In: **ICML Deep Learning Workshop**, 2015. v. 2.

KRISTAN, M.; LEONARDIS, A.; MATAS, J.; FELSBERG, M.; PFUGFELDER, R.; ZAJC, L.; VOJIR, T. et al. The visual object tracking vot 2017 challenge results. v. 1, n. 1, p. 1452 – 1459, 2017.

KRISTAN, M.; LEONARDIS, A.; MATAS, J.; FELSBERG, M.; PFUGFELDER, R.; ZAJC, L.; VOJIR, T.; BHAT, G.; LUKEZIC, A.; ELDESOKEY, A. et al. The sixth visual object tracking vot2018 challenge results. In: **European Conference on Computer Vision workshops**, 2018. v. 3, n. 5, p. 8.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: **Advances in neural information processing systems**, 2012. p. 1097–1105.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **nature**, Nature Publishing Group, v. 521, n. 7553, p. 436, 2015.

LECUN, Y.; BOTTOU, L.; BENGIO, Y.; HAFFNER, P. Gradient-based learning applied to document recognition. **Proceedings of the IEEE**, IEEE, v. 86, n. 11, p. 2278–2324, 1998.

LI, P.; WANG, D.; WANG, L.; LU, H. Deep visual tracking: Review and experimental comparison. **Pattern Recognition**, Elsevier, v. 76, p. 323–338, 2018.

LONG, J.; SHELHAMER, E.; DARRELL, T. Fully convolutional networks for semantic segmentation. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**, 2015. p. 3431–3440.

MAIA, H. d. A. et al. A mediator for multiple trackers in long-term scenario. Universidade Federal de Juiz de Fora (UFJF), 2016.

MAIA H.A.; OLIVEIRA, F. V. M. Independent selection and validation for tracking-learning-detection. International Conference on Image Processing, p. 3469–3473, 2016.

MANOCHA, P.; BADLANI, R.; KUMAR, A.; SHAH, A.; ELIZALDE, B.; RAJ, B. Content-based representations of audio using siamese neural networks. In: IEEE. **2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)**, 2018. p. 3136–3140.

NAM, H.; HAN, B. Learning multi-domain convolutional neural networks for visual tracking. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**, 2016. p. 4293–4302.

NANNI, L.; GHIDONI, S.; BRAHNAM, S. Handcrafted vs. non-handcrafted features for computer vision classification. **Pattern Recognition**, Elsevier, v. 71, p. 158–172, 2017.

NING, G.; ZHANG, Z.; HUANG, C.; REN, X.; WANG, H.; CAI, C.; HE, Z. Spatially supervised recurrent convolutional neural networks for visual object tracking. In: IEEE. **Circuits and Systems (ISCAS), 2017 IEEE International Symposium on**, 2017. p. 1–4.

OPPENHEIM, A. V.; WILLSKY, A. S.; NAWAB, S. H. **Signals &Amp; Systems (2Nd Ed.)**, 1996. ISBN 0-13-814757-4.

ORFANIDIS, S. J. **Introduction to Signal Processing**, 1995. ISBN 0-13-209172-0.

PATANE, A.; KWIATKOWSKA, M. Calibrating the classifier: Siamese neural network architecture for end-to-end arousal recognition from ecg. LOD 2018, 2018.

PERNICI, F.; BIMBO, A. D. Object tracking by oversampling local features. **IEEE transactions on pattern analysis and machine intelligence**, IEEE, v. 36, n. 12, p. 2538–2551, 2014.

POUYANFAR, S.; SADIQ, S.; YAN, Y.; TIAN, H.; TAO, Y.; REYES, M. P.; SHYU, M.-L.; CHEN, S.-C.; IYENGAR, S. A survey on deep learning: Algorithms, techniques, and applications. **ACM Computing Surveys (CSUR)**, ACM, v. 51, n. 5, p. 92, 2018.

RAFIQ, M.; BUGMANN, G.; EASTERBROOK, D. Neural network design for engineering applications. **Computers & Structures**, Elsevier, v. 79, n. 17, p. 1541–1552, 2001.

REDMON, J.; DIVVALA, S.; GIRSHICK, R.; FARHADI, A. You only look once: Unified, real-time object detection. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**, 2016. p. 779–788.

ROUT, L.; MISHRA, D.; GORTHI, R. K. S. S. et al. Rotation adaptive visual object tracking with motion consistency. In: IEEE. **2018 IEEE Winter Conference on Applications of Computer Vision (WACV)**, 2018. p. 1047–1055.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. **arXiv preprint arXiv:1409.1556**, 2014.

SZEGEDY, C.; LIU, W.; JIA, Y.; SERMANET, P.; REED, S.; ANGUELOV, D.; ERHAN, D.; VANHOUCKE, V.; RABINOVICH, A. Going deeper with convolutions. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**, 2015. p. 1–9.

TAIGMAN, Y.; YANG, M.; RANZATO, M.; WOLF, L. Deepface: Closing the gap to human-level performance in face verification. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**, 2014. p. 1701–1708.

TAO, R.; GAVVES, E.; SMEULDERS, A. W. Siamese instance search for tracking. In: **Proceedings of the IEEE conference on computer vision and pattern recognition**, 2016. p. 1420–1429.

TERPILOWSKI, M. scikit-posthocs: Pairwise multiple comparison tests in python. **The Journal of Open Source Software**, v. 4, n. 36, p. 1169, 2019.

VALMADRE, J.; BERTINETTO, L.; HENRIQUES, J.; VEDALDI, A.; TORR, P. H. End-to-end representation learning for correlation filter based tracking. In: IEEE. **Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on**, 2017. p. 5000–5008.

VO, N. N.; HAYS, J. Localizing and orienting street views using overhead imagery. In: SPRINGER. **European Conference on Computer Vision**, 2016. p. 494–509.

WANG, N.; SHI, J.; YEUNG, D.-Y.; JIA, J. Understanding and diagnosing visual tracking systems. In: **Proceedings of the IEEE International Conference on Computer Vision**, 2015. p. 3101–3109.

WANG, N.; YEUNG, D.-Y. Learning a deep compact image representation for visual tracking. In: **Advances in neural information processing systems**, 2013. p. 809–817.

WANG, Q.; GAO, J.; XING, J.; ZHANG, M.; HU, W. Dcfnet: Discriminant correlation filters network for visual tracking. **arXiv preprint arXiv:1704.04057**, 2017.

XIANG, X. A brief review on visual tracking methods. In: IEEE. **Intelligent Visual Surveillance (IVS), 2011 Third Chinese Conference on**, 2011. p. 41–44.

ZAGORUYKO, S.; KOMODAKIS, N. Learning to compare image patches via convolutional neural networks. **CoRR**, abs/1504.03641, 2015. Disponível em: <http://arxiv.org/abs/1504.03641>.