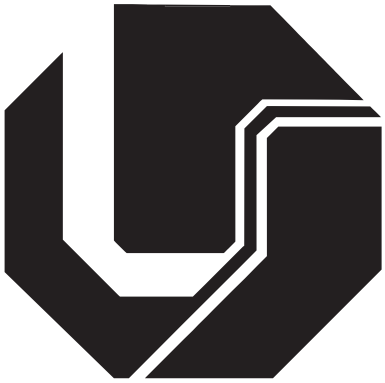
UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA



Deep Visual Tracking for Fiducial Markers: a Pilot Study in Telerehabilitation

Fernando Eduardo Resende Mattioli

Maio

2019

Deep Visual Tracking for Fiducial Markers: a Pilot Study in Telerehabilitation

Fernando Eduardo Resende Mattioli

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Uberlândia como requisito parcial para a obtenção do título de Doutor em Ciências.

Área de concentração: Processamento da Informação.

Orientador:

Prof. Edgard A. Lamounier Jr, PhD.

Coorientador:

Prof. Dr. Alexandre Cardoso

Uberlândia/MG

Maio/2019

Dados Internacionais de Catalogação na Publicação (CIP)
Sistema de Bibliotecas da UFU, MG, Brasil.

M444d 2019

Mattioli, Fernando Eduardo Resende, 1984-
Deep Visual tracking for fiducial Markers [recurso eletrônico] : a pilot study in telerehabilitation / Fernando Eduardo Resende Mattioli. - 2019.

Orientador: Edgard Afonso Lamounier Júnior.
Coorientador: Alexandre Cardoso.
Tese (Doutorado) - Universidade Federal de Uberlândia, Programa de Pós-Graduação em Engenharia Elétrica.
Disponível em: http://dx.doi.org/10.14393/ufu.te.2019.635
Inclui bibliografia.
Inclui ilustrações.

1. Engenharia elétrica. 2. Computação evolutiva. 3. Realidade aumentada. 4. Aprendizagem profunda. I. Lamounier Júnior, Edgard Afonso, 1964- (Orient.). II. Cardoso, Alexandre, 1964- (Coorient.). III. Universidade Federal de Uberlândia. Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

CDU: 621.3

Gerlaine Araújo Silva - CRB-6/1408

Deep Visual Tracking for Fiducial Markers: a Pilot Study in Telerehabilitation

Fernando Eduardo Resende Mattioli

Tese apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Uberlândia como requisito parcial para a obtenção do título de Doutor em Ciências.

Banca Examinadora:

Prof. Edgard Afonso Lamounier Jr., PhD - Orientador (UFU)
Prof. Dr. Alexandre Cardoso - Coorientador (UFU)
Prof. Dr. Eduardo Lázaro Martins Naves (UFU)
Prof. Dr. Igor Santos Peretta (UFU)
Prof. Dr. Luciano Pereira Soares (INSPER)
Prof. Dr. Márcio Sarroglia Pinho (PUCRS)

Uberlândia/MG
Maio/2019

To my son Rafael,

my endless source of inspiration.

Acknowledgements

To God, who blessed me with the opportunity to accomplish this work.

To my wife Camila and my son Rafael, for their unconditional understanding and continuous support and inspiration.

To my parents Neila and Roberto, whose examples guide me in every decision in my life.

To my brother Leandro, who always encourages and motivates me, even in the most difficult moments.

To professor Edgard, who guided me throughout this journey with kindness and wisdom, always sharing his experience and knowledge.

To my friend Daniel for his constant assistance and important contribution to this work.

To professors Alexandre, Eduardo, Igor, Luciano and Márcio for their valuable advices and contributions to this work.

To my friends and coworkers, for patiently understanding the moments of absence and for providing me with endless encouragement.

Without you, this endeavor wouldn't have been possible. My sincere and deepest gratitude to you!

If a man will begin with certainties, he shall end in doubts;
but if he will be content to begin with doubts he shall end in certainties.
- Sir Francis Bacon

Resumo

Nas últimas décadas, a evolução tecnológica trouxe grandes avanços para o campo da reabilitação humana. Dentre estes avanços, a disponibilização de serviços de reabilitação para usuários em localidades remotas - conhecida como telereabilitação - tornou possível a realização de tratamento a distância, reduzindo a necessidade de deslocamento aos centros de reabilitação. Além disso, ambientes virtuais e aumentados de treinamento possibilitaram o uso de simulações realísticas no processo de reabilitação, melhorando a experiência dos pacientes e reduzindo sua exposição aos riscos inerentes ao mundo real. Apesar destes benefícios, quando utilizada em um contexto distribuído, a tecnologia de Realidade Aumentada traz novos desafios ao processo de desenvolvimento, incluindo a integração de componentes de software de diferentes provedores. Para estas aplicações, torna-se necessário o desenvolvimento de técnicas de rastreamento mais robustas quanto aos erros de transmissão em rede. Este requisito em particular motivou a experimentação de algoritmos alternativos de rastreamento, incluindo o uso de redes neurais profundas no rastreamento visual. O papel fundamental da topologia das redes no desempenho destes rastreadores demanda a investigação de técnicas de pesquisa automática, incluindo o uso de algoritmos de computação evolutiva para esta finalidade. Neste contexto, este trabalho apresenta como contribuição um modelo de arquitetura de software para Realidade Aumentada, com foco na integração de componentes independentes para transmissão de conteúdo aumentado. Além disso, rastreadores baseados em aprendizagem profunda para detecção de marcadores fiduciais foram projetados, implementados e avaliados, apresentando desempenho superior a outros rastreadores quando aplicados a vídeos capturados por fontes remotas. Finalmente, a aplicação de algoritmos evolutivos na seleção de topologias de redes profundas é discutida, apresentando resultados competitivos quando comparados ao estado da arte.

Palavras-chave: Aprendizagem Profunda. Realidade Aumentada. Computação Evolutiva. Telereabilitação.

Abstract

In the past few decades, technology evolution has brought many improvements to the field of human rehabilitation. Among these improvements, the delivery of rehabilitation services to users at distant locations - known as telerehabilitation - made it possible to provide patients with home-based therapy, reducing the need for displacement to rehabilitation centers. In addition, the use of virtual and augmented training environments have enabled realistic simulation in the rehabilitation process, enhancing patients' experience and reducing exposure to real-world risks. Despite these benefits, when used in a distributed context, Augmented Reality technology brings additional challenges to the development process, including the integration of software components provided by heterogeneous sources. Also, the development of robust tracking techniques, less prone to detection errors resulting from network transmission is desired. This particular requirement motivated the development of alternative tracking algorithms, including the use of deep neural networks in visual tracking. When designing these trackers, the major role of network's topology selection has lead to the investigation of automatic search techniques, including evolutionary computation algorithms. In this context, this work presents as contribution a software architecture model for Augmented Reality, focusing on the reliable integration of independent software components for augmented content streaming. In addition, deep visual trackers for fiducial marker detection were designed and evaluated, outperforming other trackers when processing remotely captured video frames. Finally, the application of an evolutionary algorithm in deep neural networks' topology selection is discussed, presenting competitive results when compared to state-of-the-art methods.

Keywords: Deep Learning. Augmented Reality. Evolutionary Computation. Telerehabilitation.

Contents

List of Figures

List of Tables

List of Algorithms

List of Acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
AR	Augmented Reality
CNN	Convolutional neural network
DL	Deep Learning
DNN	Deep Neural Network
FNR	False negative rate
GA	Genetic Algorithm
LRRC	Local-rendering remote-computing
MSE	Mean squared error
PSNR	Peak Signal-to-Noise Ratio
QoE	Quality of experience
R3C	Remote-rendering remote-computing
ReLU	Rectified linear unit

RMSE	Root mean squared error
RTSP	Real Time Streaming Protocol
SGD	Stochastic Gradient Decent
SSIM	Structural Similarity
TanH	Hyperbolic tangent
VR	Virtual Reality

Chapter 1

Introduction

1.1 Motivation

In a rehabilitation context, issues associated to patient displacement to the rehabilitation centers bring additional difficulties to the rehabilitation process. Advances in computer networks technology brought the possibility of delivering rehabilitation services to patients distant from the rehabilitation centers. This technique - known as telerehabilitation [Lathan 2000] - has, among others, the objective of extending and improving patient care [Kinsella 1999]. Although home-based therapy has started to become widespread in recent years, providing telerehabilitation services with the same quality as doing in-clinic rehabilitation is still a challenge [Tayama, Hagino and Okada 2014].

The use of Virtual and Augmented Reality (AR) in health applications, specially in rehabilitation medicine, is considered a developing subject [Meer 2014]. Virtual Reality makes it possible to transport the patient to a fully simulated reality, where he is protected from many risks associated to the real world. Augmented Reality, in turn, provides the patient with better control over an augmented environment, where user interaction can be achieved in a realistic and intuitive way [Hondori et al. 2013]. Among other conclusions, research on the use of Virtual and Augmented Reality in human rehabilitation report significant enhancement on user motivation and adherence to a therapy plan, as well as a transparent way of collecting training data [Ines and Abdelkader 2011, Ortiz-Catalan et al. 2014, Popescu et al. 2000].

The development of AR systems from scratch is very challenging and requires

many skills in different domains such as image processing, 3D graphics, data acquisition among others [Didier et al. 2012]. Given the variety of available algorithms and tools, developers have to face the problem of integrating multiple technologies, provided, in many cases, by different vendors [Didier et al. 2012, Billinghurst et al. 2015]. Many task-centered AR applications are built upon specific architectures, which prevents reuse of its components, despite the similarities of many functional modules such as tracking and rendering [Bauer et al. 2001, Huang et al. 2014]. In addition, closed architectures make it harder to integrate new devices or technologies, a common requirement of many AR systems [Mossel et al. 2012].

Software architecture design has, among others, the objective of providing a higher abstraction level of the main components and their interaction in a software system [Tokunaga et al. 2003]. Quality attributes addressed by software architecture models include (but are not limited to) functionality, reliability, usability, efficiency, maintainability and portability [Dobrica and Niemela 2002]. Previous work have reported improvements on portability and integration of software components when using the layered pattern, specially in data-flow applications [Gonnet et al. 2001, Huang et al. 2014]. From a maintainability point of view, it is desirable that components should be easily replaced, whether with experimentation purposes or for adaptation to new technologies. Also, the heterogeneity of available platforms makes portability a significant issue.

In a particular class of AR applications, video frames are captured using remotely located devices. These applications include mobile edge computing [Ali and Simeone 2017], computation offloading [Lin et al. 2017] and AR distribution [Caetano et al. 2014, Chouiten, Didier and Mallem 2011, Tokunaga et al. 2003, Tokunaga et al. 2004]. In AR distribution, users at distant locations collaborate on a common task [Chouiten, Didier and Mallem 2011]. In these systems, the integration of heterogeneous components from distinct providers becomes a major challenge, given the large amount of formats, codecs and multimedia libraries available for developers [Layaida and Hagimont 2005]. In the work of Huang et al. (2013), a categorization of computation offloading applications (including AR applications) is proposed, dividing those in two main classes :

1. *Local-rendering remote-computing* (LRRC): processing tasks such as marker

tracking are executed by remote devices, and the results are returned to the local device for rendering and visualization.

2. *Remote-rendering remote-computing* (R3C): both processing and rendering tasks are executed by remote devices, and the results are returned to the local device only for visualization.

Tracking, being one of the fundamental tasks in the development of AR applications, makes it possible to overlay virtual objects in given desired positions and orientations, directly affecting users quality of experience (QoE) [Forsman, Arvo and Lehtonen 2016, Zhou, Duh and Billinghurst 2008]. Among other factors, tracking accuracy is affected by ambient lighting, distortion of captured images, light sources position, marker position, occlusion and camera resolution [Lima et al. 2017]. In addition, for distributed AR applications, the performance of marker tracking is also affected by image deterioration, resulting from compression algorithms and transmission through computer networks. The search for improvement on users QoE resulted in the development of many tracking approaches, including sensor based, computer vision based and hybrid techniques [Newman et al. 2004, Park and Park 2010, Zhou, Duh and Billinghurst 2008].

Many computer vision based trackers depend on interest point detection algorithms which are, in turn, sensitive to image deterioration [Crivellaro et al. 2015, Naqvi et al. 2015]. The search for more robust techniques motivated research on the use of many tools - including artificial neural networks (ANN) and deep learning (DL) based tools - in AR applications [Akgul, Penekli and Genc 2016, Murphy-Chutorian and Trivedi 2010, Crivellaro et al. 2015, Kim et al. 2018]. The results reported in these works indicate a competitive performance when compared to traditional detectors. However, the evaluation of these tools when processing remotely captured videos is not addressed by these works.

The hierarchical nature of digital images - in which higher level features are obtained through composition of lower level ones - makes them suitable for processing using a particular class of ANN called deep neural networks (DNN) [LeCun, Bengio and Hinton 2015]. DNN are composed by stacked layers which explore this hierarchical behavior through automatic feature extraction, each layer being responsible for a given feature level [Patterson and Gibson 2017]. With the addition of more layers and more

processing units in each layer, DNN are able to achieve significant performance in problems of increasing complexity [Goodfellow, Bengio and Courville 2016]. DNN have presented impressive results, specially in classification and regression applications, including image recognition, keypoint detection and computer vision [Assunção et al. 2018].

An important task when designing DNN is the choice of the network topology, which generaly depends on previous domain knowledge and expertise. Recently, the development of methods which minimize human interference (and therefore, the need for previous knowledge) has been discussed by researchers and practitioners [Huang et al. 2018]. Currently used approaches include random search, grid search and transfer learning. While in random search the solution space is randomly sampled, in grid search all possible combinations of a reduced solution subspace are evaluated [Bergstra and Bengio 2012]. In transfer learning, networks exhaustively trained with standard datasets are adapted to another application, by fine-tunning its weights with the target dataset [Yang, Zhao and Chan 2017].

These approaches present some limitations which narrow down their application. In pure random search, the effort of finding a good solution is not rewarded, since other solutions will be equally sampled from the search space. Grid search can be impractical if the search subspace is too large, and also subject to local optimum convergence if a non representative subspace is selected. In transfer learning, the complexity of existing and available topologies can make them unsuitable for the required application, for example if severe real-time constraints are to be satisfied. The definition of a DNN topology for a given problem can be considered an optimization problem, given the high number of parameters to be chosen [Baker et al. 2016]. Some of these parameters - including number of layers, neurons per layer, activation functions and learning algorithms - form a vast search space making exhaustive search practically impossible [Rincon et al. 2018]. In this sense, the development of methods to assist the design of new ANN topologies, with minimal intervention and limited computational resources, remains an important research topic.

Genetic Algorithms (GA) provide direct search, inspired by the Theory of Evolution, being widely applied in complex optimization problems with lots of parameters [Holland 1992, Leung et al. 2003]. GA are a subclass of evolutionary computation, which has

been successfully used to assist ANN design [Stanley and Miikkulainen 2002, Suganuma, Shirakawa and Nagao 2017]. Through genetic operators, GA are capable of exploring promising search subspaces (by combining good solutions) and also escape local optima, reducing the amount of computational resources needed when compared to other search techniques [Rincon et al. 2018, Suganuma, Shirakawa and Nagao 2017].

The environment described in the previous paragraphs motivated the investigations presented in this work. A software architecture model for distribution of AR content is presented. This architecture model includes a set of interfaces that make it possible to integrate components from different sources and roles including streaming, AR marker processing and visualization. These components include deep visual trackers, which were designed and experimented for AR marker detection and localization. Also, an evolutionary algorithm setup for use in neural network topology selection is discussed. The following section details the objectives of the present work.

1.2 Aim and objectives

The aim of this work is to present a software architecture model for AR distribution including deep visual trackers. To accomplish this, the following hypotheses, which will be further described throughout this thesis, were elucidated:

1. A layered component-based software architecture makes it possible to integrate software from distinct providers in AR distribution applications.

2. Deep visual trackers improve tracking robustness when applied to images subjected to network streaming deterioration, when compared to standard computer-vision based trackers.

3. An evolutionary topology selection procedure can be used to design competitive deep visual trackers with minimal human intervention and domain knowledge.

To answer these hypotheses, the following objectives were defined:

- To design and evaluate an architecture model, composed by a set of common interfaces, which should be implemented by streaming, AR processing and visualization software components, to provide distribution of augmented content.

- To train a set of DNN models with an AR marker tracking dataset and compare the models' performance with other trackers, when processing frames subjected to network streaming losses.
- To evaluate the performance of a genetic algorithm in DNN topology selection, when compared to other search methods such as random and grid search.

1.3 Thesis organization

This thesis is composed by seven chapters, described as follows:

- Chapter 1 describes the motivation, aim and objectives of this work.
- In Chapter 2, theoretical background is presented.
- Related work is presented and discussed in Chapter 3.
- In Chapter 4, a potential application of this work's resulting artifacts is discussed as a case study.
- Implementation details are described in Chapter 5.
- Chapter 6 discusses the results obtained with the conducted experiments.
- Finally, in Chapter 7, the conclusions of this work as well as future work suggestions are presented.

Chapter 2

Fundamentals

2.1 Introduction

In this chapter, a review of the theory and technical background of the fundamental concepts used in this work is presented.

2.2 Augmented Reality

Augmented Reality (AR) is a technology in which computer generated virtual objects overlay physical markers in real time, allowing the user to interact with the virtual objects using the markers in a natural way [Zhou, Duh and Billinghurst 2008]. In AR applications, interaction with virtual objects is achieved using real objects in a seamless way, differently from Virtual Reality (VR) systems in which the user is completely immersed in a virtual environment [Zhou, Duh and Billinghurst 2008]. A commonly accepted definition of AR was provided by Azuma (1997), describing AR as a technology which (1) combines real and virtual imagery, (2) is interactive in real time, and (3) registers the virtual imagery within the real world [Azuma 1997, Zhou, Duh and Billinghurst 2008]. Among the main domains that benefit from the use of AR technology, engineering [Engelke et al. 2013, Xin et al. 2011], entertainment [Hughes et al. 2005], education [Billinghurst and Duenser 2012, Ibanez et al. 2016], and health [Hondori et al. 2013, Lopes et al. 2014, Oliveira et al. 2016] are worth mentioning.

Developing AR applications from scratch is a challenging task, requiring skills in many different domains such as image processing and 3D graphics [Didier et al. 2012].

In a particular category of applications, called distributed AR, users at remote sites collaborate on the execution of a common task [Chouiten, Didier and Mallem 2011]. In these applications, services providing position tracking, 3D rendering, multimodal input and output are distributed across a computer network, implementing an AR pipeline from media capture to visualization [MacWilliams et al. 2003]. Therefore, the development of distributed AR applications require, in addition to the already mentioned skills, computer networks and data streaming knowledge. In this sense, the design of software architectures which simplify the integration of heterogeneous software components has been addressed in the past and is still worth investigation [Alvarado et al. 2018, Bauer et al. 2001, Bauer et al. 2003, Didier et al. 2012].

By reviewing Augmented Reality work from the period of 2008-2017, Kim et al. (2018) identified many advances in AR technology, including the improvement of many marker detection and localization techniques, which lead to the development of hybrid and learning based tracking. Despite the progress made, authors suggest the use of Artificial Intelligence (AI) and deep learning (DL) for AR tracking as an emerging research topic, not only in image processing but also to help understanding user's behavioral status. This suggestion is based on some well known DL features including the ability of DL architectures to be trained in a wide variety of capture conditions (resulting in more robust tracking) and the automatic learning of object-specific features [Garon and Lalonde 2017].

2.3 Component based software architectures

Software architecture design has, among others, the objective of providing a higher abstraction level of the main components and their interaction in a software system [Tokunaga et al. 2003]. In addition, software architectures offer solutions to important quality attributes such as generalization, modularity, reusability, maintainability and portability [Didier et al. 2012, Dobrica and Niemela 2002]. In distributed AR application development, given the variety of available tools and protocols for AR processing and network content distribution, software architecture design plays a major role on the integration of these components. From a maintainability point of view, it is desirable that components should be easily replaced, whether with experimentation purposes or for

adaptation to new technologies. Also, the heterogeneity of available platforms makes portability a key attribute.

A component based architecture allows the modularization of an application into several software components, which interoperate to achieve the application's purpose [Walton et al. 2004]. Component based architectures allows developers to partition the different aspects of a system into common areas that can be distinctly identified, specified, designed, implemented, and ultimately be reused for other potential domain applications [Wang and Qian 2005]. Software components are coherent packages of software that can be independently developed and delivered as a unit, offering interfaces through which they can be connected, unchanged, with other components to compose a larger system [Szyperski 2002]. Among the main motivations for the component pattern, Bachmann et al. (2000) highlight the following:

- Independent extensions: since components are units of extension, the component model is known to favor software flexibility, by prescribing extension rules. The component model also ensures that extensions do not have unexpected interactions, thus extensions (components) may be independently developed and deployed.

- Component markets: the integration of support services in a framework simplifies the construction of components, providing a platform upon which families of components can be designed for particular application niches.

- Reduced time-to-market: as key architectural decisions have been made when designing the component model and framework, a drastic reduction of the time it takes to design and develop new components and extensions is expected. Even if component families are not available in an application domain, the uniform component abstractions promises to reduce development and maintenance costs overall.

- Improved predictability: component models and frameworks can be designed to support the most important quality attributes for a particular application area. Since the component models express design rules that are uniformly enforced over all deployed components, global properties can be predicted for the system as a whole.

The ease of component replacement provided by component based architectures makes it possible to evaluate the performance of distinct components in a given task. In this work, two metrics described in the following sections are used to evaluate and compare the performance of distinct streaming components.

2.3.1 Image quality assessment

In the literature, methods for measuring similarity and quality of digital images have been used to quantify losses on compression algorithms, evaluate human perception and evaluate broadcast quality [Chikkerur et al. 2011, Wang and Li 2011, Moorthy and Bovik 2011]. The most used metrics for this purpose include Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity (SSIM) [Chikkerur et al. 2011]. These metrics can be used in full reference assessment, where reference images are used to quantify the quality of a set of test images.

2.3.1.1 Peak Signal-to-Noise Ratio (PSNR)

PSNR can be used as a numerical representation of image quality, and can be defined, for a grayscale reference image *f* and a grayscale test image *g* by [Hore and Ziou 2013]:

$$PSNR(f,g) = 10\log_{10}\left(\frac{255^2}{MSE(f,g)}\right) \tag{2.1}$$

where

$$MSE(f,g) = \frac{1}{MN}\sum_{i=1}^{M}\sum_{j=1}^{N}(f_{ij} - g_{ij})^2 \tag{2.2}$$

for $M \times N$ sized images.

Numerical differences between the test and the reference images lead to a higher value of the mean squared error (MSE), resulting in low PSNR. On the other hand, as the number of differences decrease, the value of MSE also decreases, leading to higher values of PSNR [Hore and Ziou 2013].

2.3.1.2 Structural Similarity (SSIM)

SSIM is used to calculate image quality through loss of correlation, luminance distotion and contrast distortion. The SSIM index is considered to be correlated with quality

perception of the human visual system, and can be defined as [Hore and Ziou 2013, Wang et al. 2004]:

$$SSIM(x,y) = [l(x,y)]^{\alpha} \cdot [c(x,y)]^{\beta} \cdot [s(x,y)]^{\gamma} \tag{2.3}$$

where $l(x,y)$ is the luminance comparison function, $c(x,y)$ is the contrast comparison function and $s(x,y)$ is the structure comparison function. The exponents α, β and γ are used as weighting factors, determining the relative importance of each component in SSIM calculation [Wang et al. 2004]. The three components can be defined as:

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.4}$$

where μ_x and μ_y are the mean luminance of reference and test images. C_1 is a constant included to avoid instability and zero denominators [Wang et al. 2004, Hore and Ziou 2013].

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.5}$$

where σ_x and σ_y are the standard deviation of luminance for both reference and test images. C_2 is also a constant, included to avoid instability and zero denominators.

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.6}$$

where σ_{xy} is the covariance between luminance of both images, and C_3 is another constant, included to avoid instability and zero denominators.

Finally, μ_x, σ_x and σ_{xy} can be defined as:

$$\mu_x = \sum_{i=1}^{N} w_i x_i \tag{2.7}$$

$$\sigma_x = \sqrt{\sum_{i=1}^{N} w_i (x_i - \mu_x)^2} \tag{2.8}$$

$$\sigma_{xy} = \sum_{i=1}^{N} w_i (x_i - \mu_x)(y_i - \mu_y) \tag{2.9}$$

where N is the window size, x_i and y_i are sample values and w is the weighting function. As suggested by Wang (2003), in this work an 11 x 11 circular symmetric

Gaussian weighting function $w = \{w_i | i = 1, 2, ..., N\}$, with standard deviation of 1.5 samples, normalized to unit sum $\left(\sum_{i=1}^{N} w_i = 1\right)$ was applied to the luminance component of images [Wang et al. 2004].

2.4 Artificial neural networks and deep learning

Artificial neural networks (ANNs) are massively parallel distributed processors, composed of simple processing units capable of storing and providing experiential knowledge [Haykin 2009]. ANNs are related to mathematical models of human cognition based on the following assumptions [Fausett et al. 1994]:

1. Information processing occurs in simple processing units called neurons.
2. Signals flow between neurons over connection links.
3. Each connection link has an associated weight (synaptic weight), which multiplies the signal being transmitted.
4. Each neuron applies an activation function to its net input (sum of weighted input signals) to determine the output signal.

In Figure 2.1, the model of an artificial neuron is presented. The input signal $x_j, j = 1, ..., m$ is connected to neuron k through synaptic weights w_{kj}, including the bias weight w_{k0}. A linear combiner (summing junction) sums the weighted inputs with a bias b_k, which increases or decreases the neuron's activation strength (v_k). Finally, the activation function is applied to the net input, resulting in the neuron's output y_k. Mathematically, this process can be summarized by equations 2.10 and 2.11 [Haykin 2009].

$$v_k = \sum_{j=0}^{m} w_{kj} x_j \tag{2.10}$$

$$y_k = \varphi(v_k) \tag{2.11}$$

Different mathematical functions are used as activation functions in ANNs. One commonly used function is the logistic sigmoid function, presented in Figure 2.2 and defined by Equation 2.12 [Goodfellow, Bengio and Courville 2016].

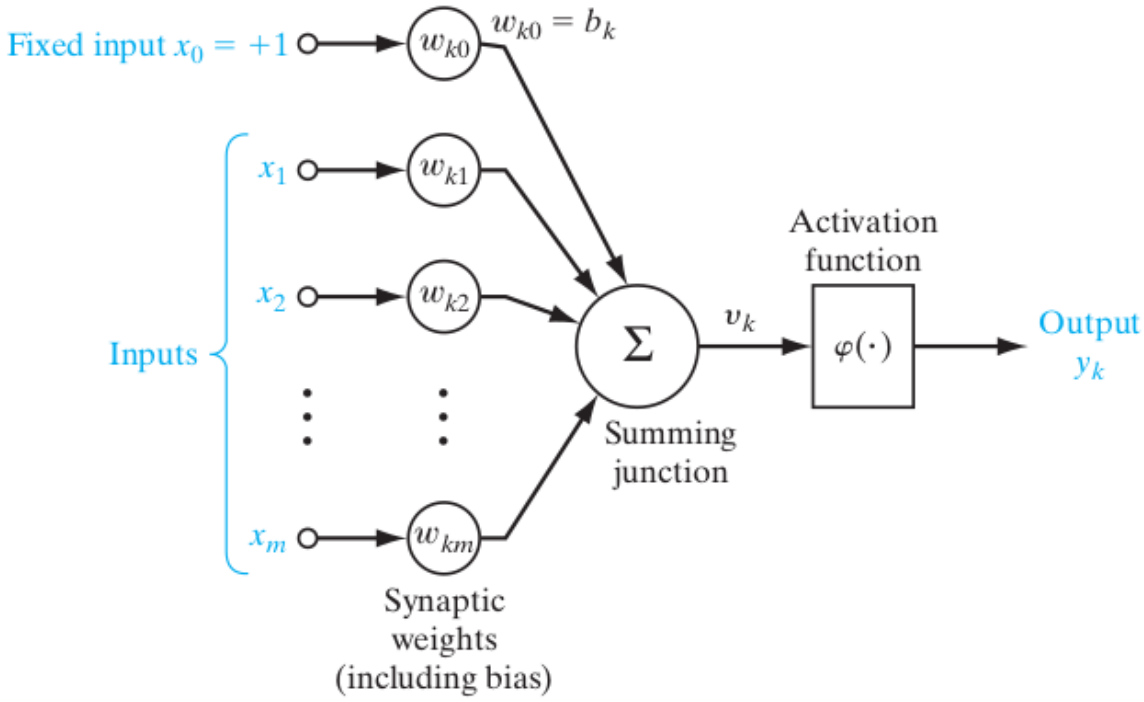


Figure 2.1: Artificial neuron model, with m inputs and 1 output [Haykin 2009].

$$\varphi(v) = \frac{1}{1 + e^{-v}} \tag{2.12}$$

The logistic sigmoid function ranges from 0 to +1. A commonly used alternative function, ranging from -1 to +1 is the hyperbolic tangent function (TanH), presented in Figure 2.3 and defined by Equation 2.13 [Fausett et al. 1994, Haykin 2009].

$$\varphi(v) = \frac{e^{v} - e^{-v}}{e^{v} + e^{-v}} = tanh(v) \tag{2.13}$$

More recently, the use of another activation function, the rectified linear unit (ReLU) has brought significant improvement on ANNs performance in many applications. The ReLU function, presented in Figure 2.4, is defined by Equation 2.14 [Goodfellow, Bengio and Courville 2016].

$$\varphi(v) = \begin{cases} v, & \text{if } v \geq 0 \\ 0, & \text{if } v < 0 \end{cases} \tag{2.14}$$

ANNs are composed of multiple layers of arranged neurons, including the input, output, and optionally a set of intermediary layers (generally referred as hidden layers).

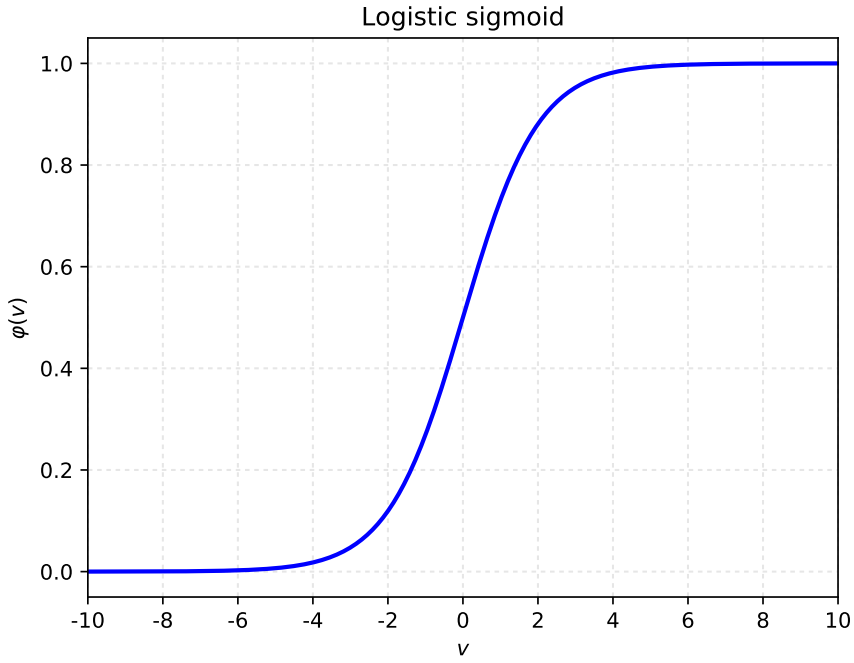


Figure 2.2: Logistic sigmoid activation function.

This arrangement of neurons, in addition to the nature of information flow, determines the network's architecture which can be grouped in three main classes [Haykin 2009]:

1. Single-layer feedforward networks: information from the input layer is directly projected in the output layer, but not in the opposite direction. No hidden layers are present.

2. Multilayer feedforward networks: information from the input layer is projected, sequentially, in a set of hidden layers, until achieving the output layer. These hidden layers act by enabling the extraction of higher-order information from the network, since the output of each hidden layer is used as input for the adjacent layer.

3. Recurrent networks: present at least one feedback loop, i.e., outputs of neurons in a given layer are used as inputs for neurons in this same layer.

Layers from which neurons are connected to all neurons of the adjacent layers are named as fully connected [Haykin 2009]. Figure 2.5 presents an example of an ANN with one hidden and one output fully connected layers.

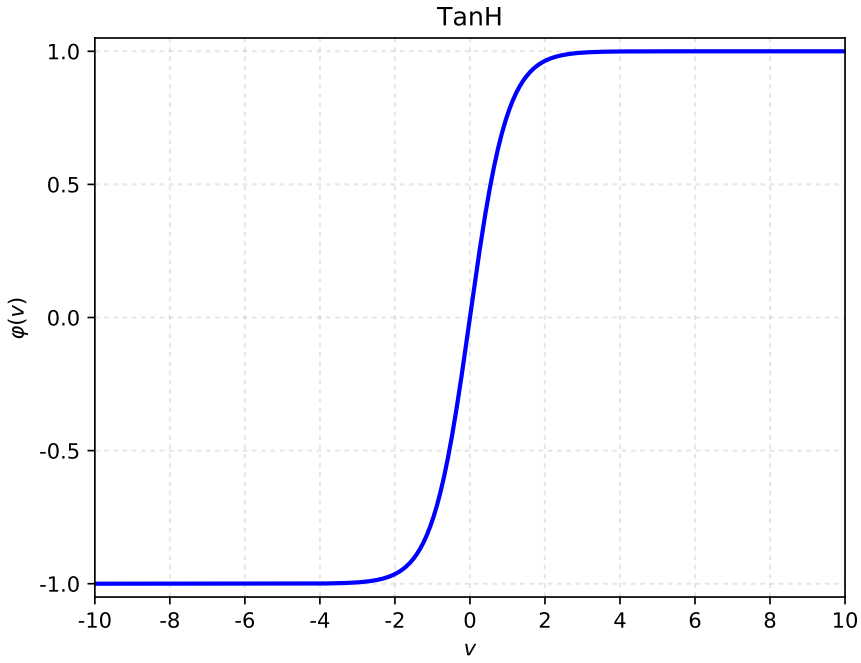


Figure 2.3: Hyperbolic tangent activation function.

2.4.1 Learning process and optimization

Artificial neural networks can acquire knowledge through different approaches. In a higher categorization, learning processes can be labeled as supervised (where a "teacher" who has additional knowledge of the environment is present) or unsupervised (without the "teacher") [Haykin 2009]. In supervised learning, a training vector containing information from the environment is presented to the learning system (ANN), which offers a response for the input data. This response is compared to the desired response (which is known only by the "teacher"), producing an error signal used to refine the learning system. This procedure is summarized in Figure 2.6.

In ANNs, the learning system refinement consists of presenting a set of training samples and updating the network weights, in order to minimize the error signal. One of the most used algorithms for this task is the error back-propagation algorithm, which is summarized as follows [Fausett et al. 1994, Haykin 2009]:

1. *Initialization*: for each layer, initialize its weights with random values from an uniform distribution.

2. *Presentation of training samples*: present the training samples to the network,

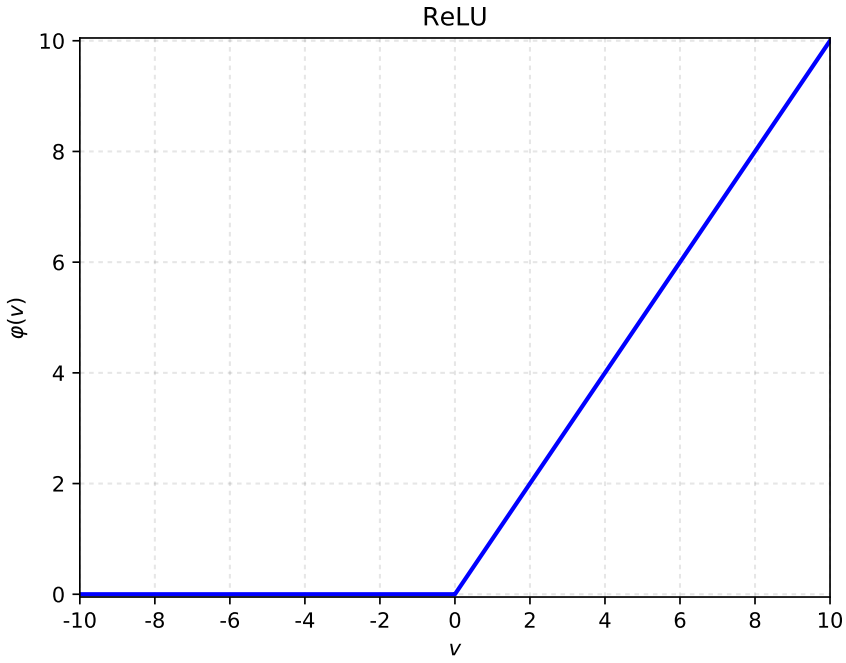


Figure 2.4: ReLU activation function.

performing the forward and backward computations described in the next steps.

3. *Forward computation*: given a training sample x, presented to the input layer with desired response (target) d, compute each layer output by proceeding forward through the network, layer by layer. The output $y_j^{(l)}(n)$ for neuron j in layer l at iteration n is calculated by Equation 2.15:

$$y_j^{(l)}(n) = \varphi\left(\sum_i w_{ji}^{(l)}(n) y_i^{(l-1)}(n)\right) \tag{2.15}$$

where $y_i^{(l-1)}$ is the output of neuron i in the previous layer $(l-1)$ and $w_{ji}^{(l)}$ is the weight of neuron j in layer l fed from neuron i in layer $l-1$. φ is the layer's activation function. For $i = 0$, $y_0^{(l-1)} = 1$ with $w_{j0}^{(l)}$ corresponding to the bias weight. The first layer ($l = 1$) has the output calculated by Equation 2.16:

$$y_j^{(0)}(n) = x_j(n) \tag{2.16}$$

For the output layer of a network with depth L, the network response to the input sample o is:

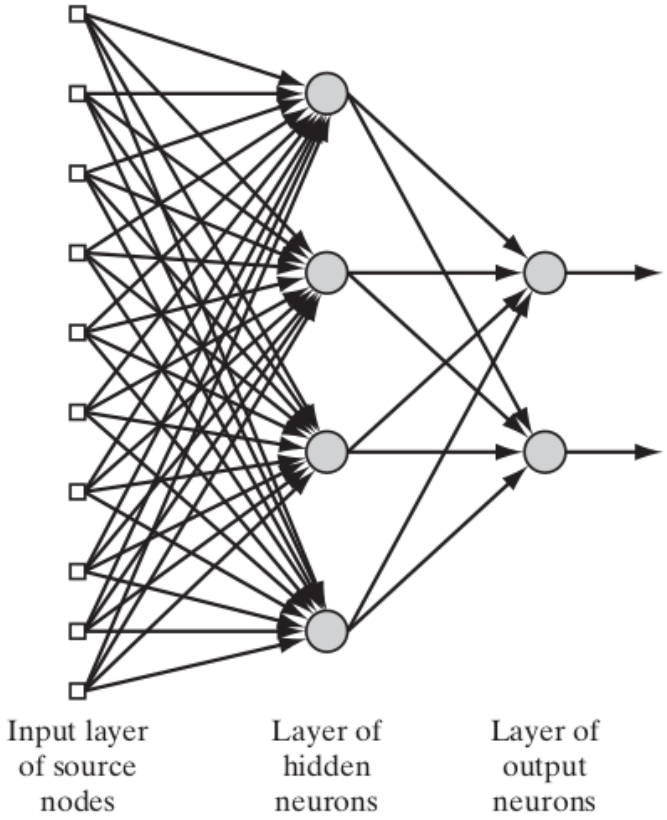


Figure 2.5: Feedforward ANN with fully connected layers [Haykin 2009].

$$o_j(n) = y_j^{(L)}(n) \tag{2.17}$$

Finally, using the desired and actual responses of the network, the local error can be calculated by:

$$e_j(n) = d_j(n) - o_j(n) \tag{2.18}$$

4. *Backward computation*: compute the local gradients of the network (δ):

$$\delta_j^{(l)}(n) = \begin{cases} e_j^{(L)}(n)\varphi_j'(v_j^{(L)}(n)), & \text{for neuron } j \text{ in output layer } L. \\ \varphi_j'(v_j^{(l)}(n)) \sum_k \delta_k^{(l+1)}(n) w_{kj}^{(l+1)}(n), & \text{for neuron } j \text{ in hidden layer } l. \end{cases} \tag{2.19}$$

where φ_j' is the differentiation of the activation function with respect to the argument. The weights of layer l are updated using the generalized delta rule:

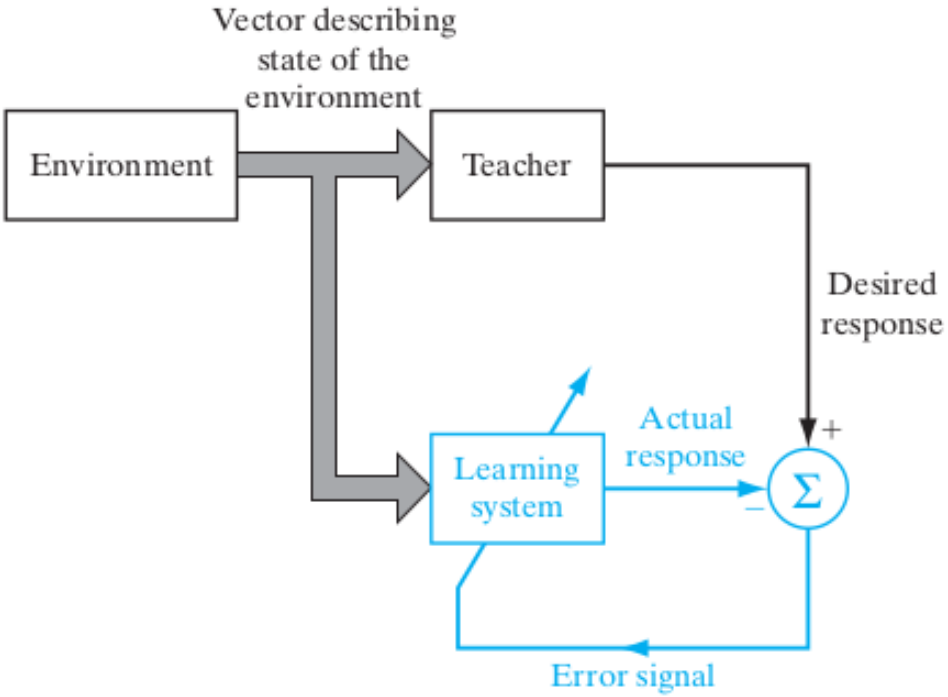


Figure 2.6: Supervised learning [Haykin 2009].

$$w_{ji}^{(l)}(n+1) = w_{ji}^{(l)}(n) + \alpha[\Delta w_{ji}^{(l)}(n-1)] + \eta\delta_j^{(l)}(n)y_i^{(l-1)}(n) \tag{2.20}$$

where η is the learning rate and α is the momentum constant. These parameters affect learning speed, determining how fast the model converges to the local optimum weight values [Sugomori et al. 2017].

Training samples are presented to the network until a stopping criterion is met. The presentation of all available training samples defines an epoch. Examples of stopping criteria include maximum number of epochs, desirable performance thresholds, early stopping (no improvement observed for a consecutive number of epochs) and validation set regularization (error increase on validation set) [Haykin 2009, Prechelt 1998].

The parameters of the ANN can be updated after the presentation of each training sample (on-line learning), after the presentation of all training samples (off-line or batch learning) or after the presentation of a subset of training samples (mini-batch learning) [Goodfellow, Bengio and Courville 2016]. In batch and mini-batch learning, parameters are updated iteratively using the gradient values (calculated, for example, using error backpropagation) through an optimization algorithm. Four well known optimization algorithms - Stochastic Gradient Decent (SGD), AdaGrad, RMSProp and

Adam - will be described in the following paragraphs.

2.4.1.1 SGD

SGD provides the update of model parameters using the average gradient on a mini-batch of m training examples. Since the learning rate is gradually decreased over time, η_k denotes the learning rate at iteration k. Algorithm 2.1 presents the SGD parameter update procedure, where $\mathbf{d}$ is the desired response (ground truth) and $\mathbf{o}$ is the actual response for a given sample. The loss function is denoted by ℓ.

Algorithm 2.1 SGD optimization algorithm [Goodfellow, Bengio and Courville 2016]

Require: Learning rate schedule $\eta_1, \eta_2, ...$

Require: Initial parameters $\mathbf{w}$

$k \leftarrow 1$

while stopping criterion not met **do**

Sample a mini-batch of m training examples $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}$, with corresponding targets $\mathbf{d}^{(i)}$

Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\mathbf{w}} \sum_i \ell(\mathbf{o}^i, \mathbf{d}^i)$

Apply update: $\mathbf{w} \leftarrow \mathbf{w} - \eta_k \hat{\mathbf{g}}$

$k \leftarrow k + 1$

end while

2.4.1.2 AdaGrad

The AdaGrad algorithm aims at performing more informative gradient-based learning by incorporating knowledge acquired in previous training iterations [Duchi, Hazan and Singer 2011]. This is achieved by scaling the learning rate of model parameters inversely proportional to the square root of the sum of historical squared values of the gradient [Goodfellow, Bengio and Courville 2016]. Algorithm 2.2 presents the AdaGrad update procedure.

2.4.1.3 RMSProp

RMSProp optimization is known to improve AdaGrad performance when applied to complex error surfaces. Instead of naively accumulating gradient values, an expo-

Algorithm 2.2 AdaGrad optimization algorithm [Goodfellow, Bengio and Courville 2016]

Require: Learning rate η

Require: Initial parameters $\mathbf{w}$

Require: Stability constant ε, typically 10^{-7}

Initialize gradient accumulation $\mathbf{r} \leftarrow 0$

while stopping criterion not met **do**

Sample a mini-batch of m training examples $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}$, with corresponding targets $\mathbf{d}^{(i)}$

Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\mathbf{w}} \sum_i \ell(\mathbf{o}^i, \mathbf{d}^i)$

Accumulate squared gradient: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$

Compute update: $\Delta \mathbf{w} \leftarrow -\frac{\eta}{\varepsilon + \sqrt{r}} \odot \mathbf{g}$

Apply update: $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$

end while

nential weighted moving average is used, reducing the contribution of the oldest gradient values by applying a decay rate ρ to the previous accumulated squared gradient [Buduma and Locascio 2017]. Algorithm 2.3 presents details of the RMSProp update procedure.

2.4.1.4 ADAM

ADAM is an optimization algorithm for first-order gradient-based optimization based on adaptive estimates of lower-order moments. In ADAM's implementation, a combination of RMSProp and momentum is presented, including bias corrections to both first and second-order moments. The ADAM optimization procedure is presented in Algorithm 2.4 [Goodfellow, Bengio and Courville 2016, Kingma and Ba 2014].

2.4.2 Convolutional neural networks

Convolutional neural networks (CNNs) are specializations of ANNs in which at least one layer uses the mathematical operation of convolution in place of the general linear combiner (summing junction) presented in Figure 2.1 [Goodfellow, Bengio and Courville 2016]. CNNs have been successfully used to process grid-formatted data,

Algorithm 2.3 RMSProp optimization algorithm [Goodfellow, Bengio and Courville 2016]

Require: Learning rate η

Require: Decay rate ρ

Require: Initial parameters $\mathbf{w}$

Require: Stability constant ε, typically 10^{-6}

Initialize gradient accumulation $\mathbf{r} \leftarrow 0$

while stopping criterion not met **do**

Sample a mini-batch of m training examples $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}$, with corresponding targets $\mathbf{d}^{(i)}$

Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\mathbf{w}} \sum_i \ell(\mathbf{o}^i, \mathbf{d}^i)$

Accumulate squared gradient: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

Compute update: $\Delta \mathbf{w} \leftarrow -\frac{\eta}{\sqrt{\varepsilon + r}} \odot \mathbf{g}$

Apply update: $\mathbf{w} \leftarrow \mathbf{w} + \Delta \mathbf{w}$

end while

such as time-series and images, in both feature extraction and classification applications [Li et al. 2018].

Convolutional layers are composed of a set of processing units (filters), each one responsible for applying a convolution kernel (a grid of numeric values similar to the synaptic weights in fully connected layers) to the layer input. The resulting output of this layer is a feature map, which will be used as input for the subsequent layers [Patterson and Gibson 2017]. Given a two-dimensional input X and a convolution kernel K with dimensions (m, n), the corresponding feature map Y is calculated using Equation 2.21 [Goodfellow, Bengio and Courville 2016]:

$$Y(i, j) = K(i, j) * X(i, j) = \sum_m \sum_n K(m, n) X(i - m, j - n) \tag{2.21}$$

In many ANN libraries, a cross-correlation function (as presented in Equation 2.22) is used in convolutional layers, since it is equivalent to Equation 2.21 with a flipped kernel while enabling more straightforward implementations [Goodfellow, Bengio and Courville 2016]:

Algorithm 2.4 ADAM optimization algorithm [Goodfellow, Bengio and Courville 2016]

Require: Step size α

Require: Decay rates for moment estimates β_1 and $\beta_2 \in [0, 1)$

Require: Stability constant ε, typically 10^{-8}

Require: Initial parameters $\mathbf{w}$

Initialize 1st and 2nd order moment variables $\mathbf{s} \leftarrow 0, \mathbf{r} \leftarrow 0$

Initialize time step $t \leftarrow 0$

while stopping criterion not met **do**

Sample a mini-batch of m training examples $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}$, with corresponding targets $\mathbf{d}^{(i)}$

Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\mathbf{w}} \sum_i \ell(\mathbf{o}^i, \mathbf{d}^i)$

$t \leftarrow t + 1$

Update biased first order moment estimate: $\mathbf{s} \leftarrow \beta_1 \mathbf{s} + (1 - \beta_1)\mathbf{g}$

Update biased second order moment estimate: $\mathbf{r} \leftarrow \beta_2 \mathbf{r} + (1 - \beta_2)\mathbf{g} \odot \mathbf{g}$

Correct bias in first order moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1-\beta_1^t}$

Correct bias in second order moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1-\beta_2^t}$

Compute update: $\Delta\mathbf{w} \leftarrow -\alpha \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}}+\varepsilon}$

Apply update: $\mathbf{w} \leftarrow \mathbf{w} + \Delta\mathbf{w}$

end while

$$Y(i,j) = K(i,j) * X(i,j) = \sum_m \sum_n K(m,n)X(i+m, j+n) \tag{2.22}$$

In Figure 2.7, an example of the convolution operation is presented. An input of size 3 x 4 is convolved with a 2 x 2 kernel resulting in a 2 x 3 output. In this example, calculation is performed only in positions where the kernel lies entirely within the image, which results in dimensional reduction when compared to the input size [Goodfellow, Bengio and Courville 2016]. When this effect is not desired, padding values can be applied to the input, in order to preserve its size. Finally, in this example, kernel has a stride of 1, meaning that it slides 1 position at a time during feature map generation. Bigger stride values lead to faster computation at the cost of greater dimensionality reduction.

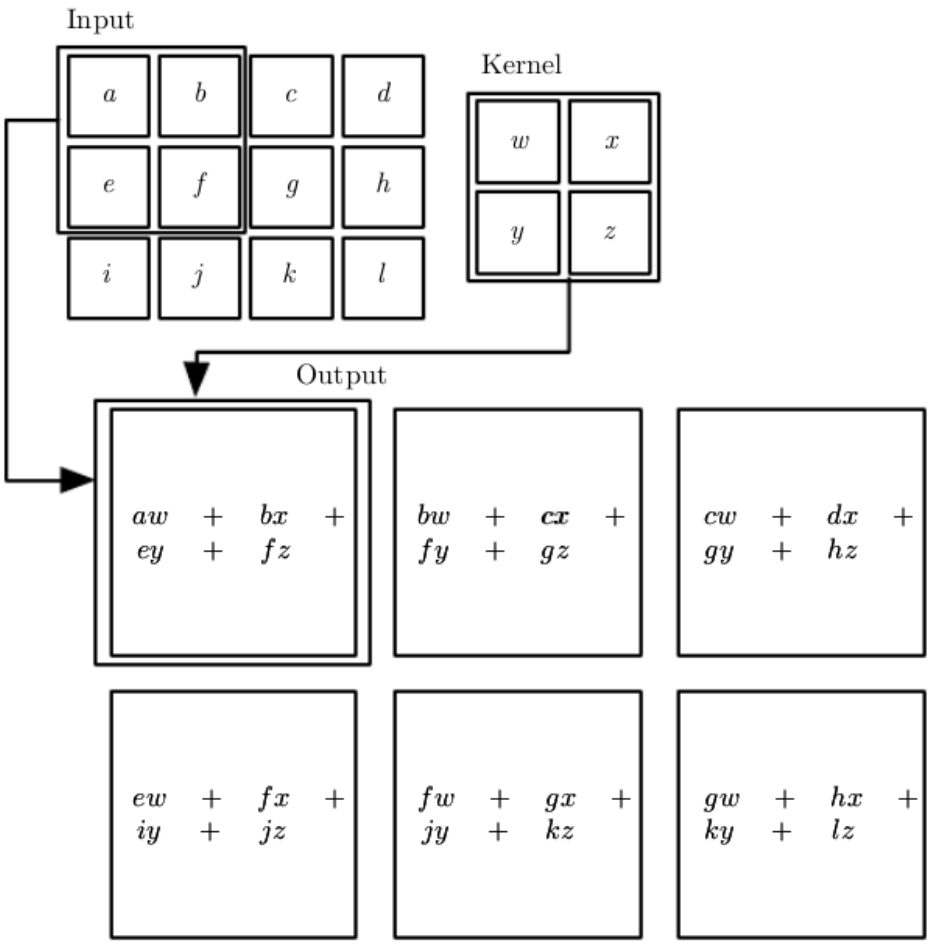


Figure 2.7: Two-dimensional convolution example [Goodfellow, Bengio and Courville 2016].

In addition to convolutional layers, CNNs often present an additional set of lay-

ers such as pooling and normalization. Pooling layers perform dimensionality reduction by replacing the feature map of a convolutional layer with a summary statistic of the nearby outputs. Most commonly used polling functions are max pooling (maximum output within a rectangular neighborhood), average pooling (average of the rectangular neighborhood) and weighted average (based on the distance of the central pixel) [Goodfellow, Bengio and Courville 2016].

Normalization layers are used to improve generalization by implementing a form of lateral inhibition of adjacent filters, creating competition for big activities amongst neuron outputs computed using different kernels. Most commonly used normalization functions are local contrast normalization and local response normalization [Jarrett et al. 2009, Krizhevsky, Sutskever and Hinton 2012].

2.4.3 Deep learning

By stacking convolutional, pooling, normalization, fully connected and other types of layers, computational models are able to process data in different levels of abstraction, defining a subset of machine learning applications called deep learning [LeCun, Bengio and Hinton 2015]. In addition, deep learning models provide automatic feature extraction capabilities, by learning the representation (features) together with the input/output mapping, differently from traditional machine learning in which the representation is defined independently of input/output mapping [Patterson and Gibson 2017, Goodfellow, Bengio and Courville 2016].

The search for improvement of deep learning models performance in most diverse applications has lead to the development of many techniques in this area. One example of these techniques is the use of dropout, which is known to reduce overfitting by randomly and temporarily removing neurons and their connections from the network during the training process [Hinton et al. 2012, Srivastava et al. 2014]. Other techniques worth mentioning are the use of unsupervised learning procedures to pre-train one hidden layer at a time [Hinton, Osindero and Teh 2006, LeCun, Bengio and Hinton 2015] and the use of evolutionary algorithms to assist or replace gradient descent based training procedures, allowing the construction of robust neural network topologies [Stanley et al. 2019]. This last topic is investigated as one of the contributions of this work, an evolutionary algorithm applied to the selection of neural network topolo-

gies trained using the backpropagation procedure.

Training a deep learning model from scratch requires two main steps: selection of the model topology and presentation of input/output data. The fundamental building blocks of a network topology are the processing layers, composed by neurons, convolutional filters, activation functions, dropout rate and spatial reductions such as pooling and stride [Dutta and Gros 2018]. These building blocks form the model hyper-parameters, and their choice is known to directly affect the network learning speed and performance [Safarik et al. 2018].

Currently used approaches for hyper-parameter search are either based on researchers skills and expertise or on a randomly exploration of the design space, often requiring much computational resources [Cai, Zhu and Han 2018, Sood et al. 2019]. In random search, the solution space is randomnly sampled, and sampled models are trained and evaluated individually, which provides paralelization capability [Bergstra and Bengio 2012]. In another search method, grid search, every possible combination of hyper-parameters in a given search subspace is used to build models which will be further indiviually trained [Bergstra and Bengio 2012]. Finally, in transfer learning, networks exhaustively trained with standard datasets are adapted to another application, by fine-tunning the weights with the target dataset, eventually prunning (i.e. removing) unessential parts of the model [Yang, Zhao and Chan 2017]. Despite the progress achieved with these approaches, a growing interest in automating neural network architecture design has lead to the development of other search methods, including reinforcement learning and evolutionary computing based methods [Baker et al. 2016, Bender et al. 2018, Real et al. 2017]. Among evolutionary computing techniques, the use of genetic algorithms has shown promising results [Stanley et al. 2019].

2.4.4 Genetic algorithms

Genetic algorithms (GA) are a set of evolutionary computing algorithms in which the solution space is explored through recombination of previous explored solutions [Russel and Norvig 2009]. In an analogy to natural selection, better solutions have a higher probability to be chosen for recombination, while bad solutions tend to be discarded [Holland 1992]. The result of this recombination process is an offspring of new solutions, in which individuals inherit different characteristics from their parents [Rincon

et al. 2018].

In standard GA, at first, a population of candidate solutions is randomly generated. Each solution is represented by a data structure (chromosome), containing the parameters to be optimized. These solutions are evaluated by a fitness function, representing the performance of the solution on the target problem with higher values being returned by the best solutions. In the next step, a selection algorithm is applied to choose existing solutions for recombination, using a fitness-proportional probability distribution [Jong 2006]. One of the commonly used selection methods is roulette wheel selection, in which better solutions have a chance of being selected which is proportional to the solution contribuiton to the total population fitness (sum of all individual's fitness). To provide better solutions with higher chances of being selected, a probability q_i is assigned to each solution s_i according to Equation 2.23, where a population of N individuals is considered [Leung et al. 2003]:

$$q_i = \frac{f(s_i)}{\sum_{k=1}^{N} f(s_k)}, \qquad i = 1, 2, \ldots, N \tag{2.23}$$

where $f(s_i)$ is the fitness of the i_{th} solution. The cumulative probability $\hat{q}_i$ for solution s_i is defined as [Leung et al. 2003]

$$\hat{q}_i = \sum_{k=1}^{i} q_k, \qquad i = 1, 2, \ldots, N \tag{2.24}$$

After randomly generating a floating-point $p \in [0, 1]$, solution s_i is chosen if $\hat{q}_{i-1} < p \leq \hat{q}_i$, given $\hat{q}_0 = 0$. The chosen individuals are then subject to the genetic operators of crossover and mutation to create an offspring of new solutions (generation). In crossover, elements of selected chromosomes are exchanged, creating new solutions inheriting characteristics of their parents. In mutation, an element of the chromosome is stochastically changed, producing a new solution. The solutions in the new generation will both be evaluated and replace the previous population, optionally keeping some of its best individuals through elitism [Leung et al. 2003, Mitchell 1998].

When applied to ANN, GA - as well as other evolutionary techniques - can be used in hyper-parameter search and/or network training, a field called neuroevolution [Stanley et al. 2019, Cuccu 2018]. In neuroevolution, each individual consists of a genotype, which is a vector of parameters representing a phenotype such as a neural net-

work [Cuccu 2018]. In direct encoding, the genotype is a weight-by-weight description of the network and its connections. In indirect encoding, the genotype contains a representation which will be used for generating the corresponding network [Stanley et al. 2019].

2.5 Final considerations

In this chapter, technical background that supported the accomplishment of this work was presented. Component-based architectures are known to favor software reuse, while enabling the integration of heterogeneous assets. For a distributed AR application, these architectures must allow the integration of specific software pieces, such as streaming and tracking components. Among these, deep visual trackers can be applied to Augmented Reality marker detection, enhancing tracking performance given their learning and generalization capabilities. The design of these trackers - more specifically, the selection of their topology - can be partially automated through the use of evolutionary computation. In the next chapter, related work will be presented and discussed.

Chapter 3

Related work

3.1 Introduction

In this chapter, related work on software architectures for distributed AR, deep visual tracking and evolutionary methods for DNN topology selection will be discussed. The references were selected by querying both Brazilian CAPES[1] and Google Scholar[2] scientific papers databases with combinations of the following keywords: “Augmented Reality”, “distribution”, “software architecture” and “software framework”, used to search work approaching AR distribution architectures; “deep learning”, “object tracking”, “object localization”, “augmented reality” and “deep visual tracking”, used for deep visual trackers search; “deep learning”, “deep neural networks”, “convolutional neural networks”, “topology selection” and “evolutionary computing”, used to filter work on DNN topology selection. For each keyword, the abstracts of the 20 most relevant references were carefully evaluated, in order to identify candidate references for further study. This procedure was then repeated, with the references sorted by publication date. After studying the full text of the chosen papers and thesis, eventually including important cited references, the papers presented in this chapter were selected to describe the state-of-the-art and the contributions addressed by the present work.

[1] http://www.periodicos.capes.gov.br/

[2] https://scholar.google.com/

3.2 Software architectures for distributed AR

The design of software architectures and frameworks for VR and AR application development have been investigated for many years, contributing significantly to many aspects such as maintainability, adaptability and technology reuse [Arcila et al. 2006, Bauer et al. 2001, Reicher 2004]. In particular, layered software architectures have proven to efficiently handle the complexity of systems by dividing the target problem into several abstraction layers, each one having its specific role and responsibility within the application [Reicher 2004, Richards 2015]. In addition, the layered architecture pattern allows compliance with many quality attributes, including independent development and modification of software components [Alvarado et al. 2018, Buschmann, Henney and Schmidt 2007].

In the work of Tokunaga et al. (2003), a software infrastructure design for distributed AR is described. The described infrastructure, named TEAR (Toolkit for Easy Augmented Reality), has the main objective of reducing the development cost of AR applications by providing a higher level abstraction of aspects such as distribution and context-awareness. To achieve this, a multimedia framework, composed by a set of multimedia components and a CORBA [Object Management Group® 2019] based communication infrastructure were designed.

Multimedia components are software entities involved in multimedia data streaming, and are divided in sources, filters and sinks. Sources are the data producers, typically interfacing with capture or media storage hardware. Filters are responsible for processing data with tasks such as AR marker detection and 3D object overlay. Finally, sinks represent the data consumers, used for displaying, storing or streaming processed data. The proposed framework provides a set of abstract classes representing sources, filters and sinks, which are extended and overridden by developers to implement the desired functionalities. Higher level components can be constructed by assembling lower level ones. Figure 3.1 presents an example pipeline in which frames captured from an input device (source) are processed by two filters, for both marker detection and 3D object superimposing. The augmented frame is then streamed to the output device (sink).

The multimedia component model used in the TEAR framework simplifies the de-

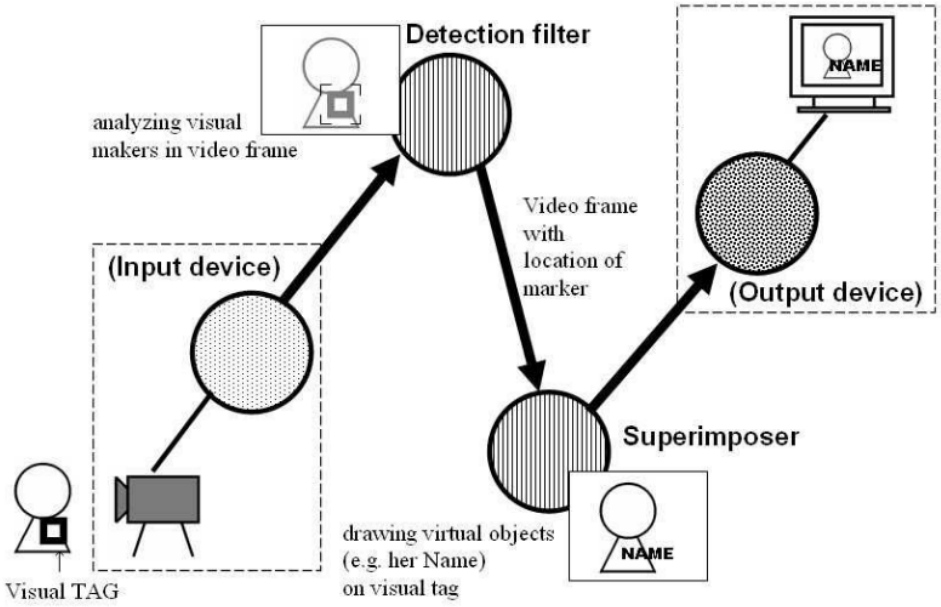


Figure 3.1: AR components and pipeline [Tokunaga et al. 2003].

velopment of complex AR applications by offering a set of abstract classes to be extended and overridden by developers to achieve the desired functionalities. The use of CORBA as a communication infrastructure enables distribution of components on multiple hosts in a standardized way, at the cost, however, of strongly coupling the framework to CORBA. Given the variety of communication protocols developed since the publication of the paper, improvements on the flexibility of the framework, especially regarding communication and distribution can be important contributions to favor its adaptation for current AR applications.

The integration of heterogeneous components in the development of AR applications is discussed in the work of Chouiten et al. (2011) and Didier et al. (2012). In the work of Didier et al. (2012), a component-based framework for AR application development is presented. The framework, named Augmented Reality Component System (ARCS), addresses software integration issues by providing a generic component model based on the signal/slot mechanism, inspired by the observer design pattern and widely used in graphical user interface libraries such as Qt [Didier et al. 2012, The Qt Company 2019].

In the ARCS framework, components are developed by extending an *AbstractComponent* class and implementing instantiation, destruction, connection management and serialization/deserialization methods. In addition, signal/slot management reflecting

the inputs and outputs of the component should also be defined. A graphical representation of a marker-based AR application built with the ARCS framework is presented in Figure 3.2.

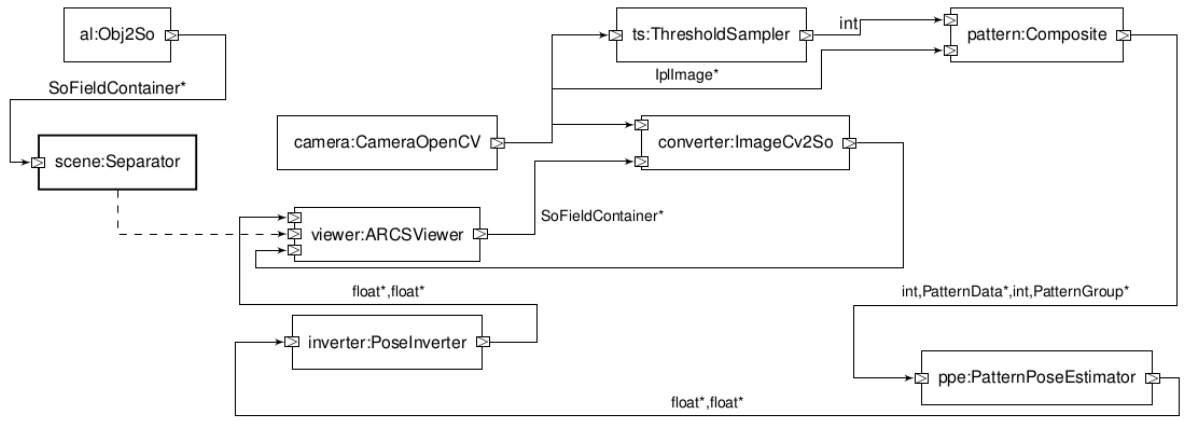


Figure 3.2: A marker-based AR application built with ARCS [Didier et al. 2012].

The camera component is responsible for capturing camera frames and dispatching them to several components, which will perform image processing tasks including marker detection and pose estimation. The 3D model is loaded by the *Obj2So* component while the *ARCSviewer* component is dedicated to the rendering of the augmented scene. In this example, a main feature of component-based development can be noticed: existing software artifacts can be integrated to the application to add new functionalities or replace existing components, as long as they are compliant or can be adapted to the component model. Therefore, the ease of integration of this artifacts is directly related to the ease of adaptation to the component model.

Chouiten et al. (2011) present a comparison of five distributed AR frameworks, namely DWARF [Bauer et al. 2001], MORGAN [Ohlenburg et al. 2004], Studierstube [Schmalstieg et al. 2002], Tinmith [Piekarski and Thomas 2003] and VARU [Irawati et al. 2008]. All frameworks are evaluated on selected criteria including ease of programming, reusability and flexibility. Ease of programming and reusability evaluation include the following aspects [Chouiten, Didier and Mallem 2011]:

- Complexity of the application model.
- Development effort required to mantain and reuse code.
- Ability to add new functionalities with minimum impact on the overall system.

Evaluating the chosen frameworks, the authors highlight the strength of component based approaches on maintainability and reusability quality attributes. Model complexity is confirmed to be balanced by the number of available features (simpler models provide less features). Flexibility is evaluated as the ability of the framework to support different hardware and software environments. The possibility of implementation in different programming languages, as well as the genericity of the component model itself are stated as important contributions to this aspect. The evaluation of these frameworks lead to the improvement of the ARCS framework, which was extended with support for distributed AR applications. In this extension, presented in Figure 3.3, intermediary components are used when communication with another machine is needed.

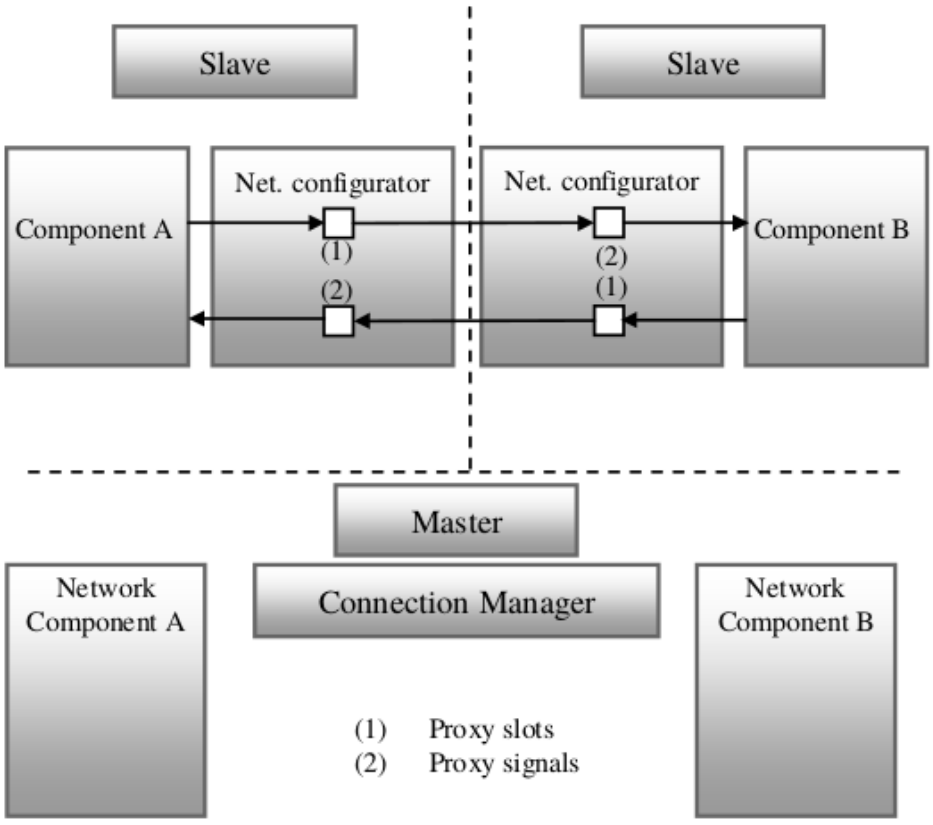


Figure 3.3: Remote connection between components in ARCS [Chouiten, Didier and Mallem 2011].

The intermediary components (network configurators) implement the proxy design pattern, including both data and connection configuration with the remote host. Each remote component is treated as a service, using the proxy signals and slots to commu-

nicate with its remote peer. A connection manager, deployed to a master host, is used to setup data flow and connections between components in slave hosts.

The work of Didier et al. (2012) and Chouiten et al. (2011) present, as main contribution, the design of a component-based AR framework that favors the integration of software from heterogeneous sources. As in the work of Tokunaga et al. (2003), the ARCS framework is strongly coupled to the CORBA architecture, which requires components to be developed following the CORBA standard. The use of more generic interfaces can favor the integration of other software components, without the explicit need to adapt them to the CORBA architecture.

In the work of Mossel et al. (2012), a common architecture for VR/AR systems is presented through a framework named ARTiFICe. The main elements of the framework's architecture, presented in Figure 3.4, include input/output devices, a computing platform, a VR/AR framework and the application itself. The framework also provides distribution capabilities, making it possible for remote users to collaborate on a common task.

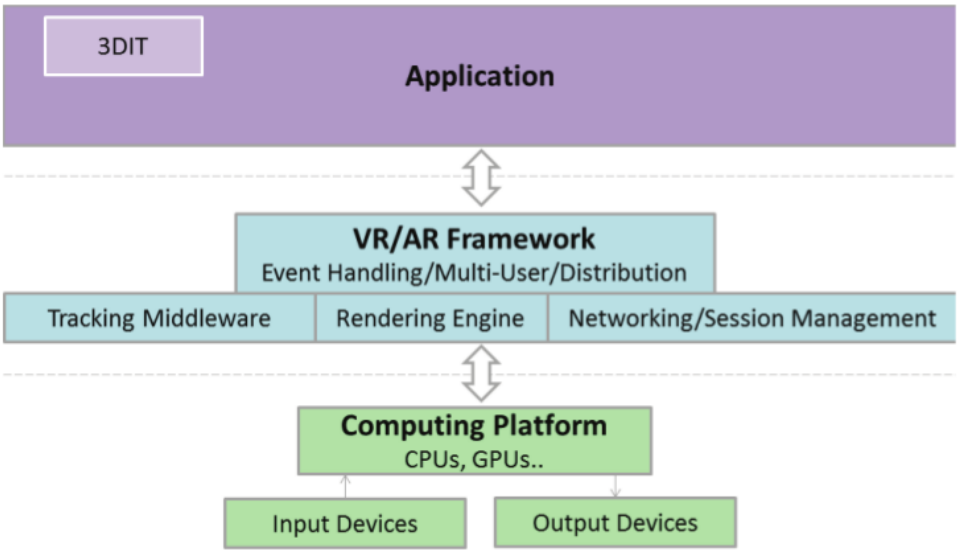


Figure 3.4: Common VR/AR system architecture [Mossel et al. 2012].

Among other contributions, the authors identify the benefits of developing loosely coupled modular frameworks for VR/AR, favoring the adaptation to novel devices and interaction techniques. Rendering and distribution functionalities, however, are strongly coupled to the Unity 3D platform. Therefore, the development of generic renderization and distribution interfaces can be important to favor the adaptation of other libraries

and tools to the framework.

After reviewing the frameworks and software architectures presented in this section, the main requirements of an AR distribution architecture favoring the adaptation of heterogeneous software components were elucidated. These requirements include the design of a flexible set of interfaces, which can be implemented by existing software assets in order to enable their integration in distributed AR applications. More details on these requirements will be presented in Chapter 5.

3.3 Deep visual tracking

Tracking has been identified as one of the main research areas in the AR literature [Zhou, Duh and Billinghurst 2008]. More recently, the use of Artificial Intelligence (AI) and especially Deep Learning (DL) techniques in tracking is described as one of the emerging topics in AR research [Kim et al. 2018]. When compared to traditional trackers in object localization tasks, DL-based trackers (also known as deep visual trackers) have shown promising results as well as valuable characteristics such as tracking robustness and automatic feature extraction [Li et al. 2018].

In the work of Sun et al. (2013) a multilevel CNN-based detector is presented, and applied in facial keypoint detection. In the presented technique, a first deep CNN takes a face image as the input and outputs rough estimates of the 2D coordinates of 5 keypoints. At the second and third levels, this estimate is refined by shallower CNNs which take as inputs smaller image patches centered on the first keypoint estimates. Figure 3.5 presents the three level cascaded CNN used for facial keypoint detection.

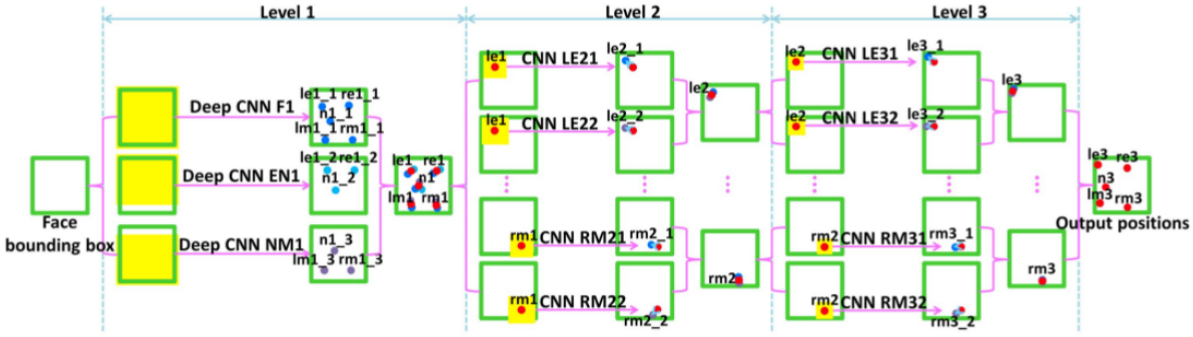


Figure 3.5: Facial keypoint detection using cascaded CNNs [Sun, Wang and Tang 2013].

Different CNNs are responsible for detecting each keypoint, using distinct input regions. While F1 takes the whole face as input, EN1 uses the eyes and nose region, and NM1 uses nose and mouth region. The inputs used in this level provide the CNNs with important contextual information, useful in a robust initial estimation. The estimates of these 3 first level CNNs is averaged to provide the image patches (yellow regions) for the second level networks, responsible for detecting left and right eye centers (LE and RE), left and right mouth corners (LM and RM), and nose tip (N). The estimates of the second level CNNs are finally refined by the third level networks, which output the final estimates of the target keypoints.

The method described by Sun et al. (2013) is reported to improve facial keypoint detection performance on the evaluated datasets, when compared to other state of the art methods and commercial software. Seven distinct neural network topologies were experimented, assessing their contribution to the overall performance. Given the accuracy of the estimates in the target problem, the application of this method to AR marker localization is an interesting research subject. Furthermore, the investigation of other network topologies can help matching response time constraints, required in the real time context of AR applications.

Crivellaro et al. (2015) demonstrate the use of convolutional neural networks on keypoint detection, applied to the detection of 3D objects. In their work, different CNNs are used to detect segments of interest and localize 2D projections of 3D control points. Three datasets containing different target objects were used to train and test the CNNs. The results obtained indicate a better accuracy on object detection when using CNNs trained with the keypoint coordinates, when compared to other evaluated methods, namely LINE-2D [Hinterstoisser et al. 2012], PWP3D [Prisacariu and Reid 2012], and LSD-SLAM [Engel, Schöps and Cremers 2014]. Also, the combination of the pose estimation of each segment of interest was reported to improve the final estimate of the object pose, making the method more robust to occlusion and background clutter. Figure 3.6 presents the tracking results for two created datasets, presenting as targets a door and a metal can.

In their work, Crivellaro et al. (2015) use two distinct CNN topologies for control point detection and projection estimation. Control point localization is setup as a detection problem, in which image patches are extracted, fed to the detection CNN and

Figure 3.6: 3D object tracking using CNNs [Crivellaro et al. 2015].

then subject to an additional processing step to elect detected parts. As reported by the authors, this procedure leads to processing times in the order of hundreds of ms. Regression DNNs with simpler architectures could improve this processing time by processing the whole image (without patch extraction) and directly predicting the 2D coordinates of the control points. However, exploration of different network architectures was beyond the scope of the authors' project.

Akgul et al. (2016) present a marker detection technique based on the classification of interest points. A well known deep CNN architecture, AlexNet, was trained to detect a set of keypoints, outperforming the ORB method, commonly used in the literature for detection of interest points [Rublee et al. 2011]. However, as pointed out by the authors, timing is still an issue, given the complexity of the AlexNet model. Therefore, the need for improvement in the tracking technique, as well as the development of simpler network architectures to increase the processing frame rate is noticed.

The analysis of the references presented in this section made it possible to define the most promising aspects of deep visual tracking applied to AR marker tracking. The tracking robustness associated with deep learning techniques can be of paramount importance in AR applications where lighting conditions, background clutter, motion blur and network streaming are known to directly affect tracking quality. In addition, it was possible to identify the need for evaluation of different network topologies in DNNs performance on the target problems, including keypoint detection. In the next section, related work on DNN topology selection will be discussed.

3.4 DNN topology selection

Recently, the use of evolutionary computing techniques in deep learning hyper-parameter optimization has been addressed by many researchers. A genetic programming method to evolve convolutional neural network architectures is presented by Suganuma et al. (2017). In the proposed method, solutions are represented by a set of building blocks, together with their connections with other blocks. Figure 3.7 presents an example of the proposed genotype and its corresponding phenotype.

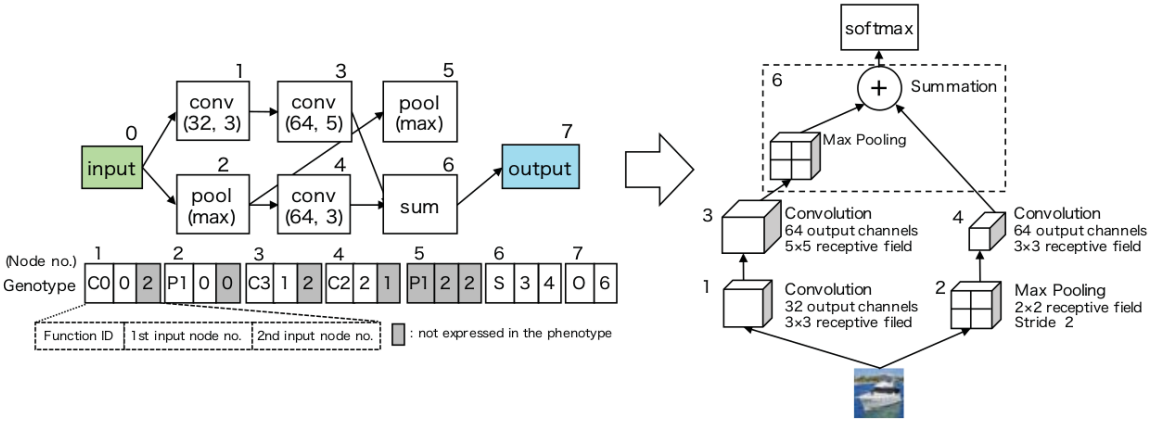


Figure 3.7: DNN building blocks and connections [Suganuma, Shirakawa and Nagao 2017].

The proposed encoding makes it possible to represent complex flow graphs through building blocks and connections. Mutation is applied by changing a specific node's block or connection. Solutions are evaluated based on their accuracy on a test dataset, after being trained with backpropagation.

When applied to the CIFAR-10 dataset [Krizhevsky and Hinton 2009], the proposed method was able to find competitive network topologies, compared to state-of-the-art classifiers. However, as pointed out by the authors, training each model from scratch, for a limited number of epochs has a considerable computation cost. Alternatives to prevent the waste of resources with potentially bad solutions, such as early stopping or performance prediction should bring important improvements to the presented method.

In the work of Assunção et al. (2018), genetic algorithms are used to evolve the structure and parameters of a deep neural network. The encoding schema, named DENSER (Deep Evolutionary Network StructurEd Representation), is based on Dy-

namic Structured Grammatical Evolution (DSGE), allowing mapping and representation of complex structures and their constraints. Figures 3.8 and 3.9 present examples of genotypes and phenotypes constructed using the proposed representation.

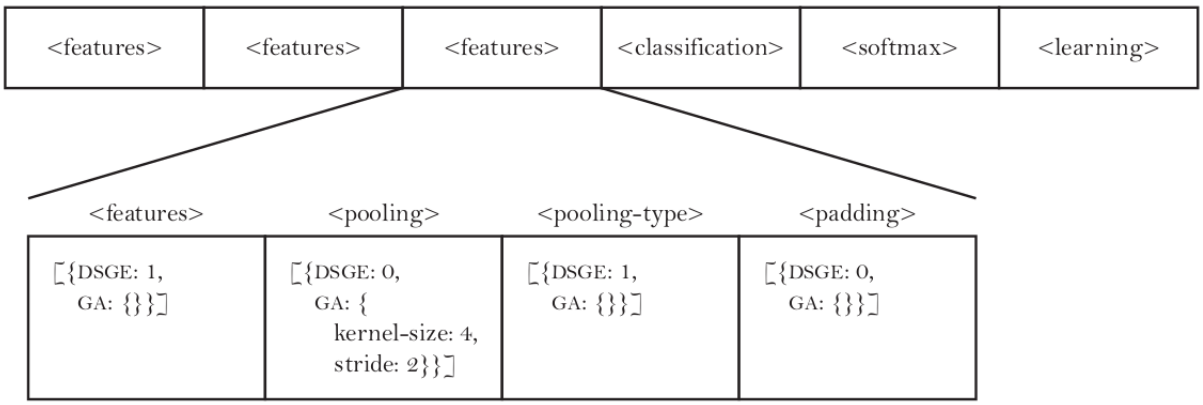


Figure 3.8: Genotype of candidate solution encoding a CNN [Assunção et al. 2018].

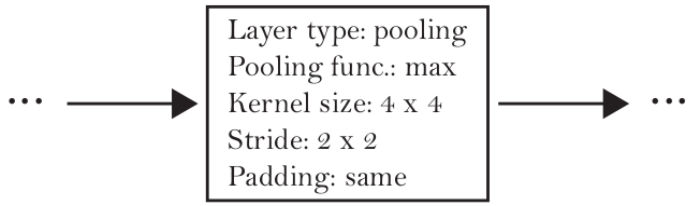


Figure 3.9: Corresponding phenotype [Assunção et al. 2018].

The proposed representation makes it possible to encode complex DNN structures, including convolutional layer parameters (named after *features* in the chromosome), fully connected layer parameters (named after *classification*), and also output and learning parameters. Crossover is performed either by generating cut points or by swapping genes between parent solutions. Mutation is performed by adding, removing or replicating layers from the solution's descriptor. The CIFAR-10 dataset, used to demonstrate the proposed technique, is partitioned into training, validation and test sets. Each solution (i.e. DNN) is trained for 10 epochs, and the accuracy on the validation set is used as the solution's fitness. After evolving a population of 100 solutions for 100 generations, the best solutions are retrained for 400 epochs and evaluated on the test set.

The work of Assunção et al. (2018) presents an encoding scheme that favors adaptation to other domains and makes it easy to configure and apply genetic operators to candidate solutions. The conducted procedure can be adjusted to other datasets and

applications, such as localization and keypoint detection. Despite the evaluation for only a small number of epochs, training all models from scratch can require a prohibitive amount of computational resources, depending on dataset characteristics such as frame size and number of available training samples. As mentioned previously, this issue can be addressed by early stopping the training process when model performance improvement for the validation set is not observed for a number of consecutive epochs. In addition, fitness predictors can be used to discard potentially bad solutions, directing computational resources for more promising models.

Rincon et al. (2018) present the use of evolutionary optimization and convolutional neural networks in a cancer diagnosis application. A fixed topology of 3 convolutional and 1 fully connected layer is used, and the hyper-parameters of these 4 layers (filters, kernel size, pooling window and number of neurons) are encoded in the chromosome. A population of 50 individuals, in which each solution is trained for 1000 iterations is setup.

When compared to the state-of-the-art, the CNNs obtained by Rincon et al. (2018) outperformed the reference classifiers, presenting promising performance in the target application. However, the proposed fixed chromosome can drastically limit the search space exploration. Therefore, the investigation of more flexible representations would bring important contributions to the obtained results.

The results reported in the works described in the present section confirm the benefits of the application of evolutionary computing in deep learning hyper-parameter optimization. However, the computational cost of these approaches is still high, given the number of solutions that must be evaluated by the optimization algorithms. Therefore, techniques to optimize the use of computation resources in this class of applications remain an important area of inquiry [Ullah and Petrosino 2016]. An example of such a technique is presented in Smithson et al. (2016). In their work, a method based on design space exploration to optimize DNN hyper-parameters is presented. The proposed method uses an ANN to predict the performance of a given topology. Together with Pareto efficiency, this prediction prevents the waste of computational resources by not evaluating potentially bad solutions. Inspired by the contributions discussed in this section, this work demonstrates the simultaneous use of a flexible encoding scheme and fitness prediction in DNN topology selection applied to digital image processing,

including an AR marker tracking application.

3.5 Final considerations

After reviewing the aforementioned research projects, three main research contributions presented in this thesis were identified: 1) the need for an architecture model for distributed AR, with the main requirement of making it possible to integrate and experiment components from distinct vendors with ease; 2) the need for development of more robust AR tracking components (for example, deep visual trackers), capable of maintaining an acceptable performance when processing images deteriorated by network transmission; 3) the need for development of deep learning hyper-parameter optimization techniques, suitable for application in image processing tasks. These contributions were investigated throughout this work, which presents a software architecture model for distributed AR and addresses the development of deep visual trackers for AR marker detection and localization, using GA in model hyper-parameters optimization. Table 3.1 summarizes the main contributions of related work to the experiments and techniques presented in the next chapters of this thesis.

Table 3.1: Related work summary and contributions.

Author(s)	Distributed software architecture	Augmented Reality	Deep visual tracking	Evolutionary hyper-parameter optimization
Tokunaga et al. (2003)	✓	✓		
Chouiten et al. (2011)	✓	✓		
Didier et al. (2012)		✓		
Mossel et al. (2012)	✓	✓		
Sun et al. (2013)			✓	
Crivellaro et al. (2015)		✓	✓	
Akgul et al. (2016)		✓	✓	
Suganuma et al. (2017)				✓
Assunção et al. (2018)				✓
Rincon et al. (2018)				✓
Mattioli (2019)	✓	✓	✓	✓

Chapter 4

Case study

4.1 Introduction

In this chapter, an application of the software architecture model and components proposed in this work will be discussed. This application will be used as a case study of the present work, and by the time of this writing, is under validation by researchers from the Virtual and Augmented Reality Group and the Laboratory of Assistive Technology from the Faculty of Electrical Engineering of the Federal University of Uberlândia.

4.2 Computer assisted rehabilitation

In the past few decades, technology evolution has brought great improvement to the field of human rehabilitation. Among the main applications, one can highlight the development of Computer Assisted Rehabilitation Systems. The widespread use of these systems brings into discussion some aspects of the rehabilitation process itself, such as methods standardization, assessment metrics and general usability. Research on the use of computers as assistive tools in human rehabilitation have been conducted since the early 1990s [Reni et al. 1991, Anand and Guha 1995]. Benefits associated to the use of computers in the rehabilitation process can be divided in two main categories: (1) provision of assistance and feedback to the patient; (2) acquisition and provision of information to the therapist.

One common example of a computer assisted rehabilitation system is the implementation of training environments. These environments aim to simulate real ones,

accurately, reproducing the challenges encountered by users in their daily activities. One of the main advantages of this approach is user safety, since the simulation environment do not expose the patient to real world risks.

In many cases, standardized exercises are executed and data collected from rehabilitation systems can be used to asses the performance of individuals in relation to groups. However a common requirement of many assistive rehabilitation systems is exercise customization, to address patient specific needs [Tetteroo et al. 2015]. In these cases, patient's performance is evaluated in relation to previous results, obtained by himself.

4.3 Virtual/Augmented Reality and rehabilitation

Virtual Reality is an advanced user interface, having as its main features 3D visualization, spatial navigation and interaction with environment's components [Cardoso and Lamounier 2006]. In many situations, access to a physical training environment can be restricted by some constraints such as time and space. When such limitations are faced, Virtual Reality can be used to extend availability of training environments, also bringing the training exercises to the patient. Although the use of technology to support rehabilitation is not a new phenomenon, improvements in hardware and cost reduction have recently increased the interest in the use of Virtual Reality and gaming technology for "Virtual Rehabilitation" [Powell, Powell and Simmonds 2014].

One example that illustrates the use of Virtual Reality in human rehabilitation is the development of virtual training environments. These environments provide, by their nature, the possibility of extending the application's availability to a large number of users, at the same time. Among other aspects, modeling techniques and real-time response play a major role on the degree of immersion provided by the training system.

Augmented Reality makes it possible to optimize the user experience, since it creates an altered reality without losing benefits of the physical setting [Wiederhold and Wiederhold 2014]. In the study conducted by Ines and Abdelkader (2011), the use of Augmented Reality techniques has proven to enhance user motivation and acceptance. Also, by augmenting a dedicated physical environment, it is possible to customize it, according to each user specific needs.

Moreover, with the development of Augmented Reality distribution techniques, it can be noticed that the same training environment can be shared by multiple users in different locations, at different times [Silva 2011]. This can be very useful for some applications, specially for wheelchair users training: a physical environment with some real world difficulties, such as ramps and obstacles, can be augmented with some virtual objects, that can be used as virtual obstacles or checkpoints. This customization would help adapting the training environment to the user's needs.

4.4 Telerehabilitation

As defined by Deutsch [Deutsch, Lewis and Burdea 2007], "telerehabilitation is the provision of rehabilitation services (evaluations and interventions) at a distance by therapists at a remote location". In the past few years, telerehabilitation technology has been developed seeking to address some issues related to classical in-clinic treatment [Postolache et al. 2011, Pani et al. 2014]. Among these issues, one can highlight the availability of the physical environment (and equipment) [Brennan et al. 2011] and the time and expenses associated to patients and therapists displacement [Dhurjaty 2004].

Telerehabilitation technology can be used to improve existing rehabilitation protocols. Since many standard treatments include high number of repetitive exercises, patients are required to perform additional exercises at home, without the supervision of a therapist [Ortiz-Catalan et al. 2014]. In these cases, telerehabilitation software can be used to provide the therapist with valuable feedback. As stated by Ortiz-Catalan *et al.* (2014), "the monitoring of quality and quantity of home-based rehabilitation interventions can potentially have a large positive impact on the quality and adherence to long-term treatments and rehabilitation outcomes".

4.5 Distributed AR and powered wheelchair users telerehabilitation

Burdea (2004) describes a computational system used to evaluate or train patients at a distance, essentially, composed by 3 main modules:

a. Exercise software, running on the patient's station.

b. Remote monitoring software, running on the therapist site.

c. A database and remote graphics capability, used for patient's medical data.

In this context, considering the concepts explored in the previous sections of this chapter, an architecture for a computer assisted telerehabilitation system applied to the training of powered wheelchair users is proposed. This architecture is composed mainly by three different environments:

1. A training environment, in which the exercises protocol will be executed.
2. A control environment, from which the user remotely controls the wheelchair.
3. A supervision environment, in which a health professional can follow the exercises executed by the patient and access performance data.

An outline of the main components of the proposed solution is shown in Figure 4.1.

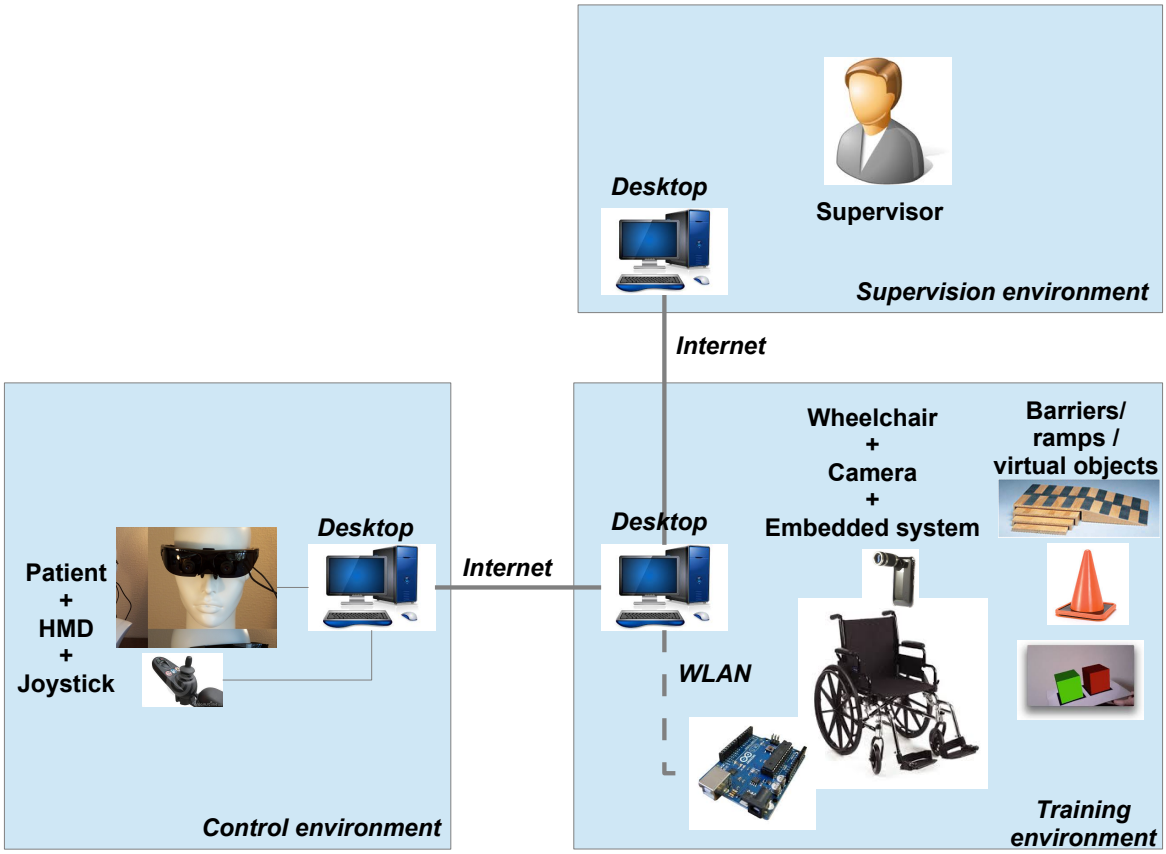


Figure 4.1: System's architecture outline

4.5.1 Training environment

The proposed training environment is composed by an unmanned wheelchair (a wheelchair without human control), a set of physical barriers/ramps, a set of Augmented Reality markers and a desktop computer. The physical barriers and ramps will be used during exercises. Additionally, virtual barriers will be positioned over the Augmented Reality markers, according to the configuration proposed by the therapist.

A camera is integrated to the wheelchair, to provide remote clients (patients and therapists) with the visual feedback. These clients have, in their respective environments, a vision of the augmented environment (physical and virtual barriers), from the wheelchair point-of-view.

The wheelchair is controlled by an embedded system, which receives control signals from the desktop. The desktop, in turn, receives control signals from the patient - located in the control environment - and sends these signals to the embedded system. Control signals are received through the Internet and sent through a WLAN network.

4.5.2 Control environment

In the proposed control environment, the patient is able to issue control comands to the remote wheelchair by using a joystick. The visual feedback (captured in the training environment, from the wheelchair point-of-view) is provided through a monitor in this environment.

A desktop computer is used to provide communication with the patient: control commands are collected from the joystick and further sent to the training environment; images from the training environment, containing the physical and virtual barriers, are received through the network and projected on the head-mounted display.

4.5.3 Supervision environment

On telerehabilitation applications, it is desirable that therapists can follow the execution of the exercises [Burdea 2004]. In the presented architecture, this can be achieved in the supervision environment. In this environment, the therapist uses a desktop computer to follow the execution of the exercises by his patient. Also, the therapist would be able to configure the Augmented Reality markers, in order to customize the set of

exercises according to patient's development in the training process.

4.5.4 Training process

The training process itself consists on the execution of a training protocol. In the training protocol, the therapist establishes some series of exercises, to be executed by the patient. Each exercise should provide some assessment metrics, used with 2 main purposes:

a. To evaluate the patients' progress during a time window.

b. To evaluate individual performance, comparing it to group statistics.

On in-site training environments, used on wheelchair users' training, the patient executes the proposed exercises, controlling a wheelchair in order to avoid the positioned barriers. To use this same approach on a telerehabilitation system, some adaptations must be considered. The patient will be remotely controlling an unmanned wheelchair, performing the same exercises that he/she would do in-site. Thus, a two-way communication channel is needed: the patient should be able to send remote movement commands to the wheelchair and the wheelchair, in turn, should provide the patient with visual feedback. In the training environment presented on Figure 4.1, an embedded controller is responsible for handling interaction between the wheelchair and an workstation, through an wireless network. The workstation is responsible for communicating with other two environments: control environment and supervision environment.

Augmented Reality technology provides the supervisor (therapist) with the possibility of adding virtual objects to the physical training environment. These objects can then be used to extend and customize training sessions according to patient's needs, and also to enhance user experience and motivation during the execution of the exercises.

4.6 Final considerations

This chapter presented a computer assisted rehabilitation application, which motivated the investigation discussed in the rest of this thesis. As a potential application of the

main assets produced by the present work, the main requirements of the system described in this chapter were used to guide the design of the software architecture and tracking techniques explored in the next chapters. The implementation and evaluation of this application with potential users is being conducted by partner researchers from the Virtual and Augmented Reality Group and the Laboratory of Assistive Technology from the Faculty of Electrical Engineering, Federal University of Uberlândia. By the time of this writing, preliminary results of these projects can be found in [Caetano et al. 2014], [Caetano et al. 2015] and [Caetano et al. 2018].

Chapter 5

Implementation

5.1 Introduction

In this chapter, implementation details of the techniques and experiments described in this work will be presented.

5.2 Architecture model

Software components to provide distribution of augmented content should address some high level requirements such as:

1. Provide an interface with capture devices.
2. AR marker tracking (detection and pose estimation).
3. Superimposing 3D images over the detected AR markers.
4. Augmented media streaming between hosts.

Additionally, in order to favor adaptability and flexibility (given the variety of available tools for AR processing and content streaming), it is desirable that these components implement standard interfaces. To match these requirements, a layered software architecture, presented in Figure 5.1, was designed.

In the media server side, the capture interface is responsible for providing access to the capture hardware (e.g. camera device). AR processing is divided in three tasks, namely marker recognition, pose estimation and 3D overlay. As each of these tasks

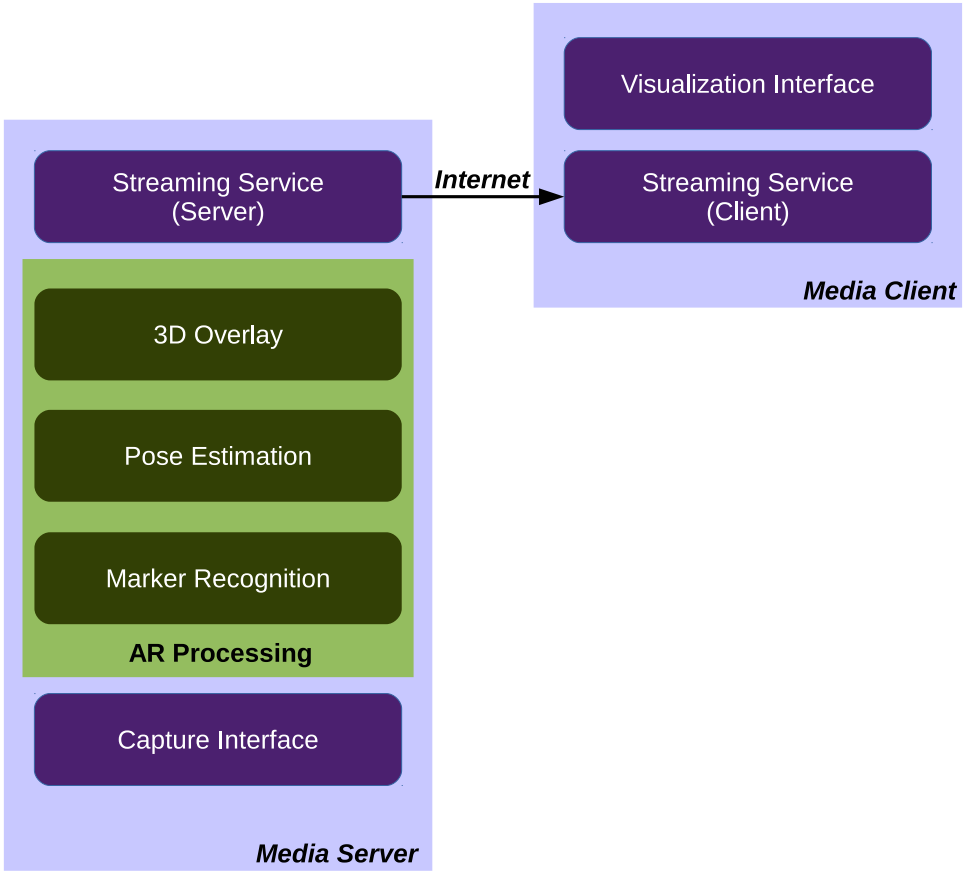


Figure 5.1: Software architecture layers.

can be implemented by different frameworks, each task was designed as a layer in the main architecture. After AR processing and 3D overlay, the augmented media is sent by the streaming service to the media client. A client endpoint for the media server is implemented within the media client. Once received, the augmented media is finally presented to the user through the visualization interface.

In order to evaluate the proposed architecture, two distinct implementations of its components were developed. On the first implementation, the OpenCV [OpenCV team 2019] library was used for media capturing and AR processing. The streaming service was developed using the libVLC [VideoLAN organization 2019] streaming library. In the second implementation, the WebRTC [WebRTC 2019] library was used for media capturing and streaming. AR processing was implemented using the JSARToolKit [ARToolKit 2019] library.

5.2.1 Proof of concept: OpenCV + libVLC

The integration of OpenCV and libVLC technologies made it possible to capture videos from the camera, perform AR processing and stream the resulting augmented media to a media client, using the RTSP protocol [Schulzrinne, Rao and Lanphier 1998]. Each layer in the architecture model was implemented as a set of classes and interfaces, through which communication between adjacent layers are established. The implemented components are presented in Figure 5.2.

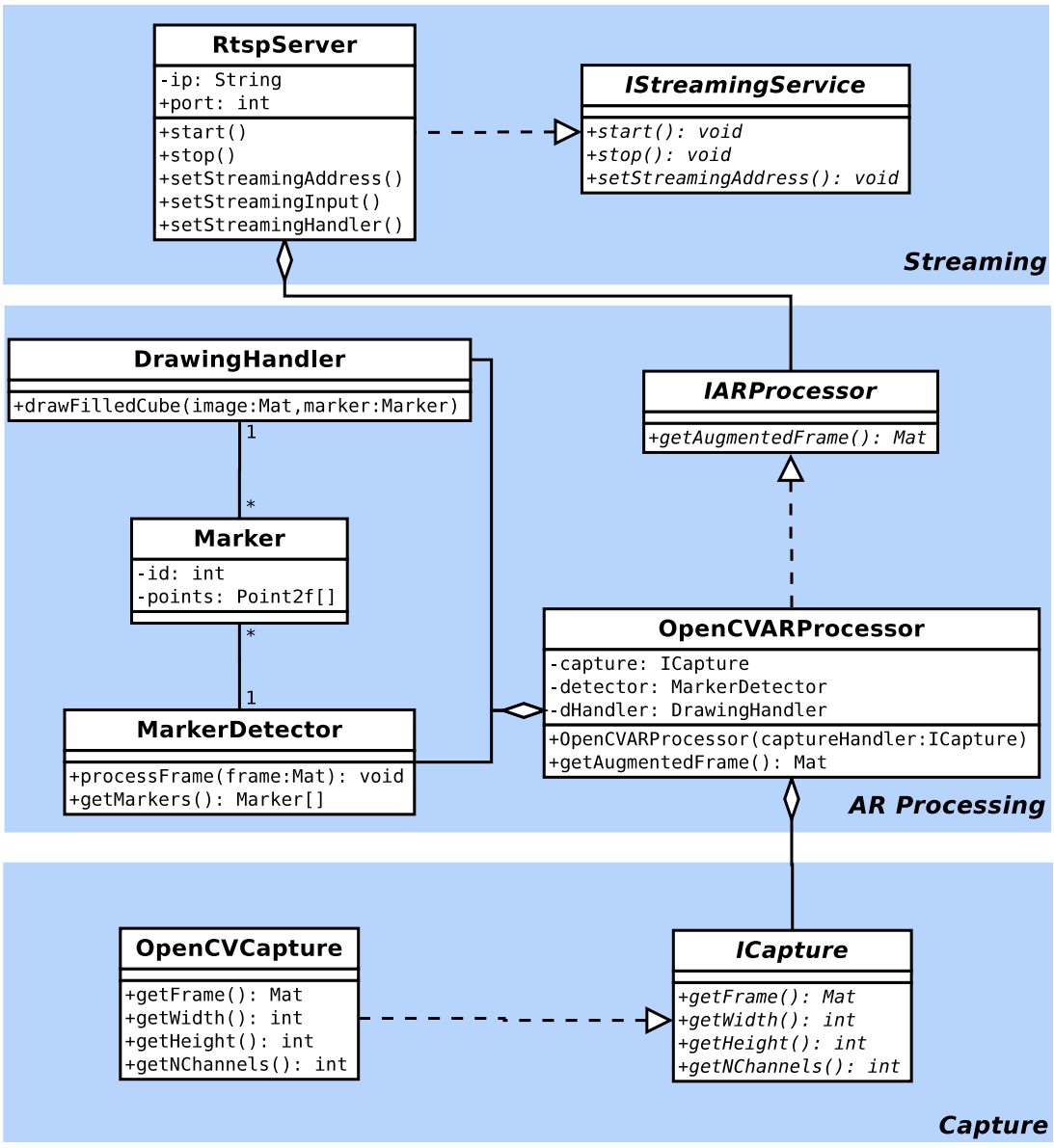


Figure 5.2: OpenCV/libVLC integration classes.

The *OpenCVCapture* class provides the interface with the capture device, through the OpenCV library. Gathered frames are then passed to the AR processor, imple-

mented by the *OpenCVARProcessor* class. Marker detection and frame augmentation are performed by the *MarkerDetector* and *DrawingHandler* classes. The following workflow - proposed by Baggio *et al.* [Baggio 2012] - was implemented: 1) convert the luminance channel of the input image to grayscale; 2) perform static binary threshold operation; 3) detect contours; 4) search for possible markers; 5) detect and decode markers; 6) estimate marker 3D pose.

After marker pose estimation, the *DrawingHandler* class draws a 3D virtual object over the detected marker, and the augmented frame is sent to the streaming layer. In this layer, the *RtspServer* class performs encoding and streaming of the augmented frames, through an RTSP channel.

5.2.2 Proof of concept: WebRTC + JSARtoolKit

In the second proof of concept, the browser was used as the implementation platform. Capture and streaming interfaces were built using the WebRTC framework. The JSARToolKit library was used to provide AR functionalities, such as marker detection and pose estimation. 3D drawing was implemented using the three.js [three.js 2019] library. Figure 5.3 presents an outline of the main classes used in this implementation.

The *CameraService* class provides access to the camera frames, through an HTML 5 video element. Camera opening is an asynchronous event, since it requires user authorization. Therefore, in order to keep track of the global application workflow, the *Observer* design pattern was used [Gamma et al. 1994]. In this case, the *CameraService* class keeps track of a list of subscribers, which will be notified of any state change, such as camera opened, access denied, etc. On the AR processing layer, the *ARService* class provides marker detection and scene renderization. The auxiliary *DrawingHandler* class is used to draw 3D objects over the detected markers. Similarly, the *Renderer* class is used for both foreground and background renderization. Finally, on the streaming layer, the *StreamingService* class is responsible for establishing the peer connection with the receiver side. To achieve this, the *SignalingService* class provides basic signalization. Since targeted signaling messages are asynchronous, this class also implements the *Observer* pattern, as well as the *Singleton* pattern, to ensure a single entry point to this service [Gamma et al. 1994].

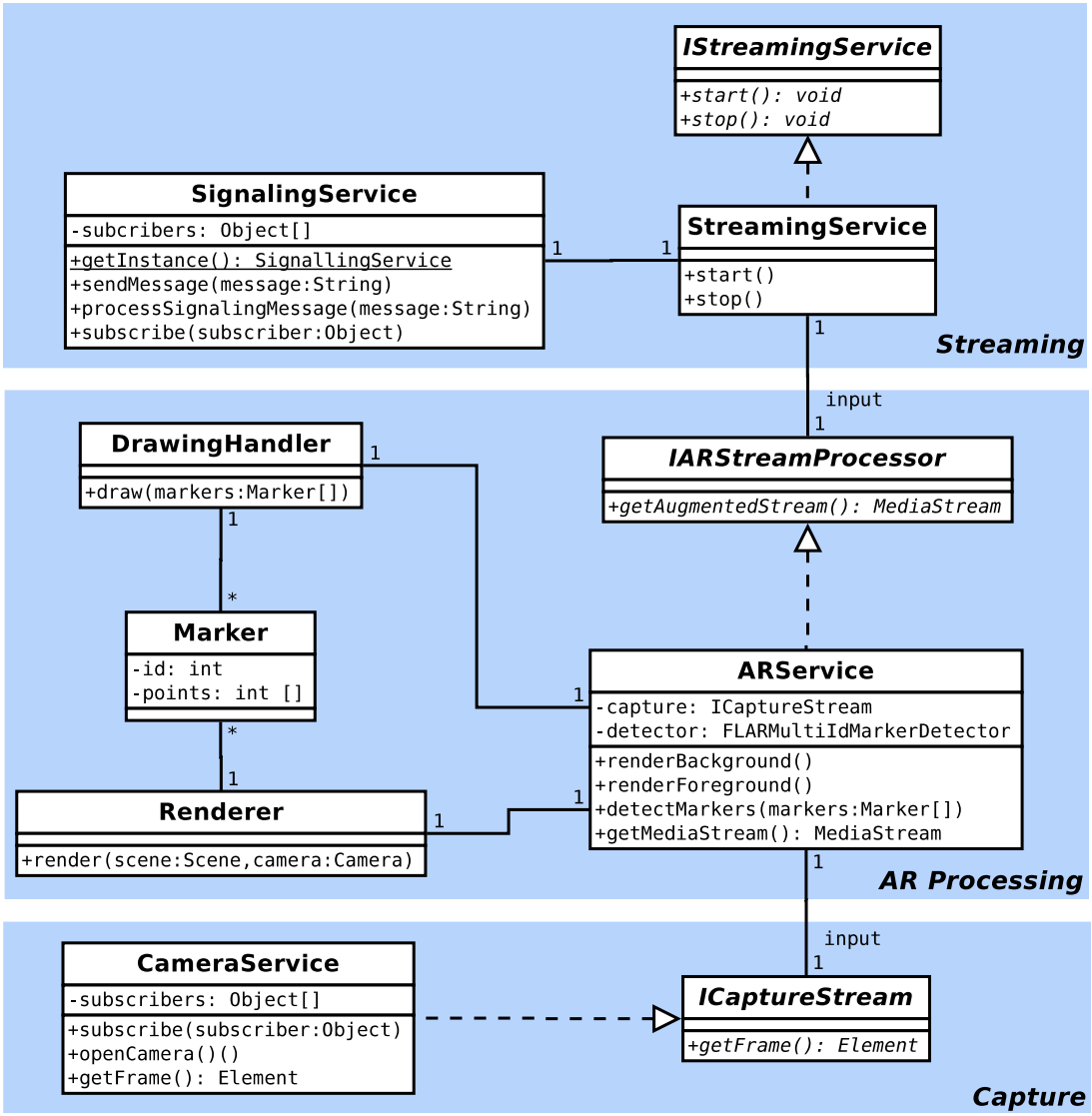


Figure 5.3: WebRTC/JSARToolKit integration classes.

5.3 Architecture evaluation

In the conducted experiments, a 1.8 GHz CPU, 8 GB RAM computer running Debian Linux 9 and Mozilla Firefox 60 was used as the media server. The media client was deployed to a 1.6 GHz CPU, 4 GB RAM computer running Debian 9 and Mozilla Firefox 60. Both computers were connected to a local wired network using Ethernet cables.

Reference videos were produced, aiming at evaluating the architecture's performance under distinct controlled conditions. These videos contain fiducial markers - similar to the AprilTag markers [Olson 2011] - in different combinations of number (1, 3 or 5) and condition (fixed or rotating). Rotation axes were defined as presented in Figure 5.4 (a). Figures 5.4 (b, c, d) present example frames of the rendered videos. For

further reference in this work, static marker videos were labeled as S*N*, with *N* being the number of markers in the frame. Rotating marker videos were labeled as R*AN*, with *N* markers rotating over the *A* axis.

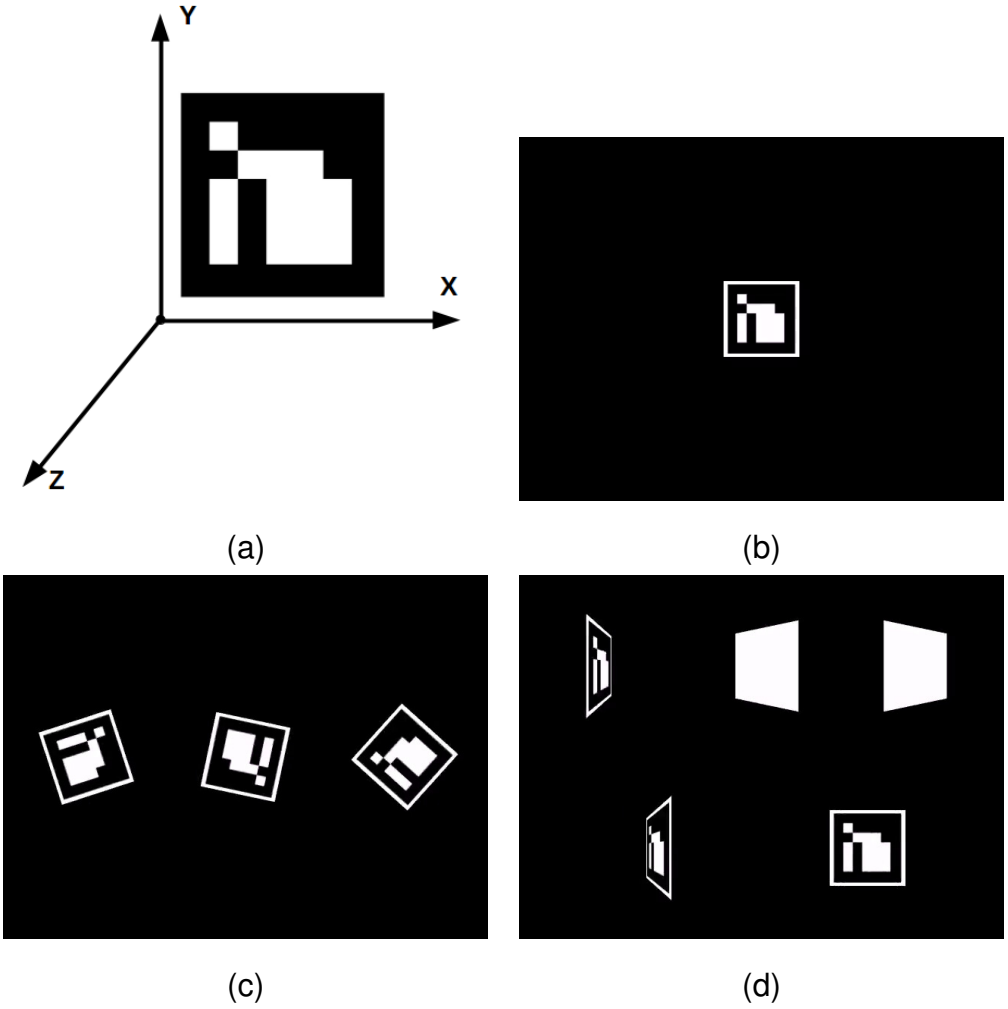


Figure 5.4: Reference videos: (a) Marker rotation axes. (b) S1: single static marker. (c) RZ3: three rotating markers (z axis). (d) RY5: five rotating markers (y axis).

A camera simulation software[1] was used to feed these videos to the media server. The server was started and kept running for 5 minutes. Every 2 seconds, a sample of CPU and memory usage was collected, using the *ps* tool [The UNIX and Linux Forums 2019]. Meanwhile, on the client's side, the number of received frames were collected, as well as the corresponding timestamps.

To examine the processing/transmission delays presented in the proposed architecture, a delay model was defined. This model considers time delays related to frame

[1] https://sourceforge.net/projects/webcamstudio

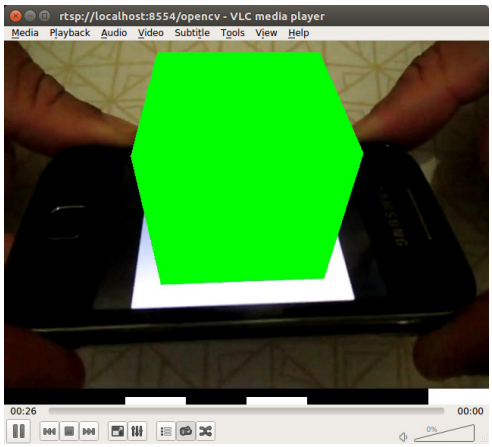
capture, AR processing and network transmission, including video encoding/decoding delays. Delay measurement was achieved by adding instrumentation code at specific points of the proposed architecture model, as described in Table 5.1.

Table 5.1: Delay measurement.

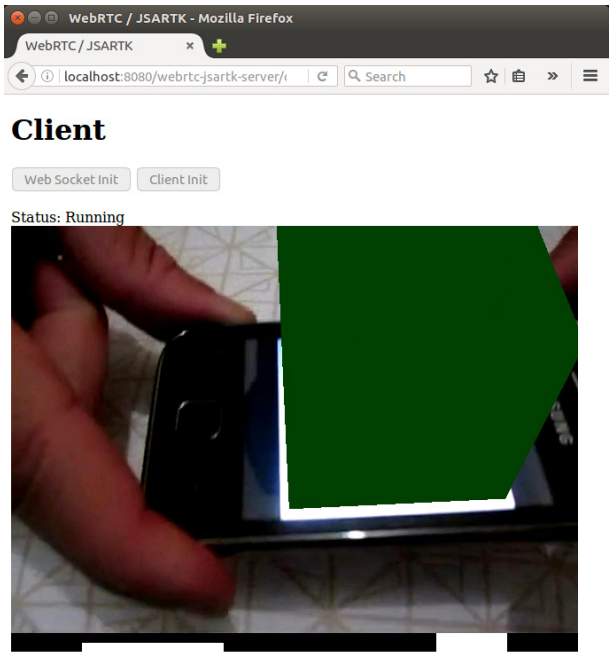
Variable	Description
t_1	Timestamp after frame capture
t_2	Timestamp after frame processing
t_3	Timestamp after frame receiving
Δt_p	Processing delay ($t_2 - t_1$)
Δt_t	Transmission delay ($t_3 - t_2$)

The presented timestamps were collected by setting up the following experiment: first, a reference video in which an AR marker is moved and presented in different positions was recorded, aiming at simulating real world operating conditions. The reference video was recorded for 50 seconds, at 30 FPS (320x240 resolution), resulting in about 1500 frames. This video was then subject to a processing filter, which encoded each frame with an unique binary identification line, presented in Figure 5.5. The camera simulation software was used to feed the encoded video to the media server. When a new frame is ready in the capture interface, timestamp t_1 is stored, as well as the frame identification (corresponding to the binary line). The same procedure is executed in the AR processing layer, to collect timestamp t_2. When a frame is available at the client, timestamp t_3 is stored, together with frame identification. Since the recorded video was fed to the media server once (without loop), the gathered data was further grouped by frame identification, and used to calculate average processing and transmission delays. Accurate measurement of transmission delays requires synchronization between server and client's clocks. To validate the measurement technique, this experiment was conducted executing both server and client in the same host, therefore, using the same clock.

In distributed AR applications, users' quality of experience is directly related to the quality of delivered video. Significant losses on delivered image frames - related to compression algorithms or dropped network packages - can compromise user experience. Therefore, the need for measuring video quality arises. In this work, two met-



(a) OpenCV/libVLC.



(b) WebRTC/JSARToolKit.

Figure 5.5: Frames with binary identification lines.

rics widely used in the literature and described in Chapter 2 were chosen to quantify delivered video quality: Structural Similarity (SSIM) and Peak Signal-to-Noise Ratio (PSNR) [Chikkerur et al. 2011]. Both metrics require comparison of target images with the corresponding references. While SSIM presents a percentage of similarity, related to user perception of the image, PSNR is directly related to the mean squared error, and is expressed in dB [Hore and Ziou 2013].

Since both SSIM and PSNR are image-related, to quantify the streamed video's quality one needs to process this video in a frame-by-frame basis. In addition, each frame received by the client should be matched with the corresponding reference frame, sent by the server. To achieve this, the luminance channel of the binary line encoded video was used, in loop, as the input for the media server. During streaming, sample frames were randomly collected, until a number of 1000 samples were gathered at the client's side. Finally, the collected frames were decoded and matched (client sampled frames with corresponding server sampled frames). Among the matching frames, 200 samples were randomly chosen, for further analysis.

5.4 Deep visual trackers

In order to develop a deep visual tracker for AR, the test video described in the previous section was used, with the objective of simulating real world operating conditions. This video was transmitted through a computer network using the streaming components of the two proofs of concept described earlier in this chapter. The NetEm [Kerrisk 2019] simulator was used to emulate different network conditions, including distinct package delays (0, 20, 40, 60, 80 and 100 ms) and package loss rates (0, 5, 10 and 15%). To identify video frames after transmission, the binary code (black and white regions) at the bottom of each frame was used, as presented in Figure 5.6. After transmission, sample frames were collected in the media client, and used for further analysis. Finally, 20 datasets using different combinations of delay, package loss and transmission protocol were produced.

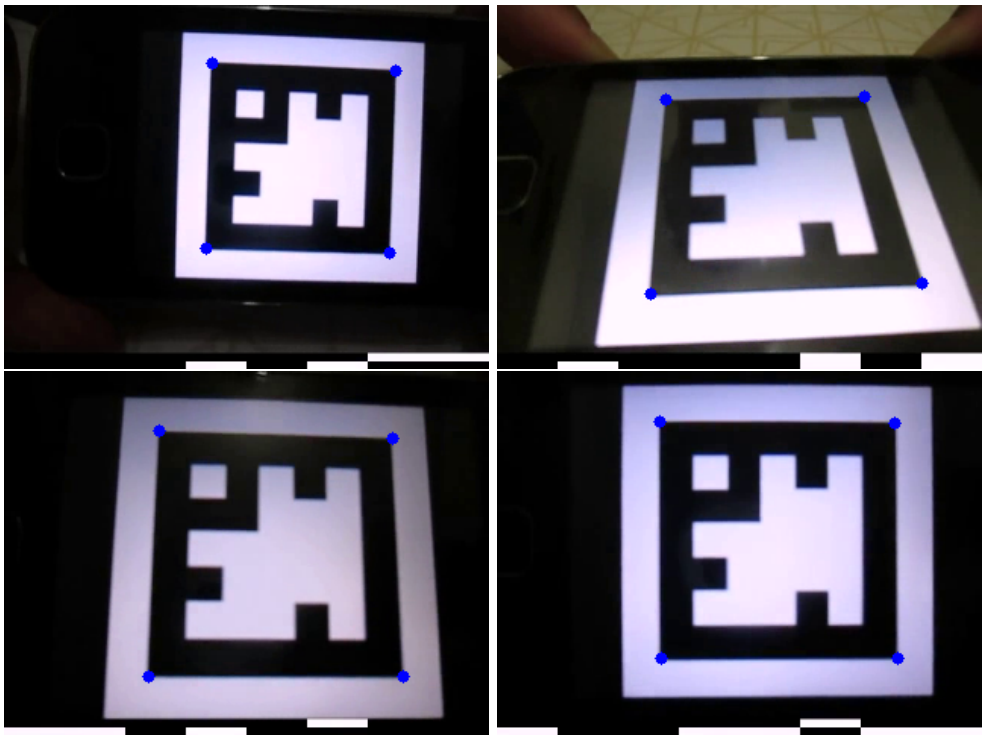


Figure 5.6: Coded frame examples and keypoints.

The designed deep visual trackers were implemented using the Caffe framework [Jia et al. 2014] and the C++ programming language. In a first approach, different network topologies were experimented, following a grid-search based procedure [Welchowski and Schmid 2016, Albelwi and Mahmood 2017]. In this procedure, input and

output layers are defined to match the dataset configuration. For hidden layers, three reference network architectures were evaluated: the first architecture is composed only by fully connected layers. The second architecture is composed by convolutional, pooling and normalization layers. Finally, the third architecture is composed by convolutional, pooling, normalization and fully connected layers. The rectified linear unit (ReLU), logistic sigmoid and hyperbolic tangent (TanH) activation functions were experimented [LeCun, Bengio and Hinton 2015]. The acronyms used in this work to represent the network building blocks are presented in Table 5.2.

Table 5.2: Acronyms.

Acronym	Description
FC(n)	Fully connected layer, with n neurons.
FCR(n)	Fully connected layer, with n neurons, activated by ReLU.
FCL(n)	Fully connected layer, with n neurons, activated by logistic sigmoid function.
FCT(n)	Fully connected layer, with n neurons, activated by TanH.
C(n,k,s)	Convolutional layer, with n filters, kernel size k and stride s.
CR(n,k,s)	Convolutional layer, with n filters, kernel size k, stride s, activated by ReLU.
MP(k,s)	Pooling layer (max pooling), with kernel size k and stride s.
LRN(k)	Normalization layer (local response normalization), with kernel size k.

Two distinct datasets were used for training and testing the different topologies of neural networks evaluated in this work. The search procedure has the objective of finding the network topologies with minimal detection (or localization) errors for the proposed problems. The procedure was divided in two steps, described in the following paragraphs:

Step 1 - Depth Search: in this step, each model was trained until no error reduction for 5 training epochs is observed in the validation set, for a maximum of 500 epochs. From the initial setup defined empirically and presented in Table 5.3, new layers (or layer blocks) were added to the network, while error reduction was observed. For the fully connected layers, the ReLU, logistic and TanH activation functions were evaluated.

For convolutional layers, four layer blocks were experimented: 1) convolutional layer; 2) convolutional layer + ReLU; 3) convolutional layer + ReLU + pooling layer and 4) convolutional layer + ReLU + pooling + normalization layer.

Table 5.3: Initial setup.

Architecture	**Configurations**
FC	FC(8)
	FCR(8)
	FCL(8)
	FCT(8)
CONV	C(8,11,4)
	CR(8,11,4)
	CR(8,11,4) - MP(2,2)
	CR(8,11,4) - MP(2,2) - LRN(5)
CONV+FC	C(8,11,4) - FC(8)
	CR(8,11,4) - FC(8)
	CR(8,11,4) - MP(2,2) - FC(8)
	CR(8,11,4) - MP(2,2) - LRN(5) - FC(8)

Step 2 - Tunning: in this step, the best configurations of each model were selected to be used as references. The other network parameters (number of neurons, kernel size, stride, etc.) were modified by increasing or decreasing the value of the parameter in the neighborhood of the reference value. If the evaluated configuration presented a decrease in the error when compared to the reference, the latter was replaced by the evaluated configuration, until no decrease in the validation error was observed.

In this step, all evaluated networks were trained for a maximum of 1000 epochs, early stopping when no improvement in the validation set was observed for 50 consecutive epochs. The Adam optimization method was used, with learning rate 0.001 and the other parameters configured as proposed by [Kingma and Ba 2014]. Figure 5.7 summarizes the described procedure.

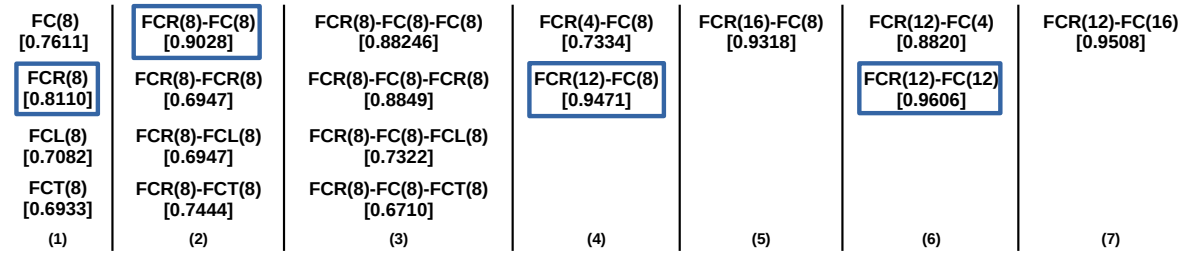


Figure 5.7: An example of the topology evaluation procedure: the accuracy of each model is presented between brackets; the best configurations (reference models) are highlighted. (1) In the first step, four distinct models are evaluated and the reference model is defined. (2) A new layer is added to the best model of the previous step, obtaining a new reference model. (3) A new layer is added to the reference model, without improvement in the accuracy. (4 - 7) Local search is conducted, modifying the number of neurons in each layer and replacing the reference model when improvement is observed. Finally, the best model obtained in this example is FCR(12)-FC(12).

5.5 Deep neural network topology selection

With the objective of improving the performance of the deep visual trackers described in the previous section, the application of an evolutionary algorithm in network topology selection was experimented. In these experiments, four topology search methods (described later in this section) were evaluated, in a first step, with the MNIST and CIFAR-10 datasets [Krizhevsky and Hinton 2009, LeCun et al. 1998]. This evaluation was conducted to assess the efficiency of the technique in well know datasets, with reference metrics available in the literature. Once the technique was validated, it was applied to the AR marker tracking dataset.

For the MNIST dataset, 1024 samples were randomly extracted for training, while 256 samples (distinct from the training set) were chosen as the validation set. With the CIFAR-10 dataset, 2000 samples for training and 500 samples for test were used. To provide fair comparisons, all search methods used the same data. Model performance was measured as the test accuracy, i.e. the number of test patterns correctly classified divided by the total number of test patterns, as presented in Equation 5.1.

$$ACC(\%) = \frac{n_C}{n_C + n_W} \times 100 \tag{5.1}$$

where n_C is the number of test patterns correctly classified and n_W is the number of wrong classifications.

In order to represent model complexity, the number of connections in the model was used. These two metrics - test accuracy and number of weights - were used to evaluate a solution with respect to the Pareto front, in a multi-objective optimization problem: maximize the test accuracy and minimize the number of weights.

A customized genetic algorithm was developed using the C++ programming language, and a personal computer without GPU acceleration was used to train the models. In all evaluated search methods, each solution is represented by two stacks of layers, the first one containing convolutional layers and the second containing fully connected layers. Each layer is presented as a string descriptor, as shown in the examples of Figure 5.8. To facilitate parsing these descriptors, elements are separated by semicolon characters.

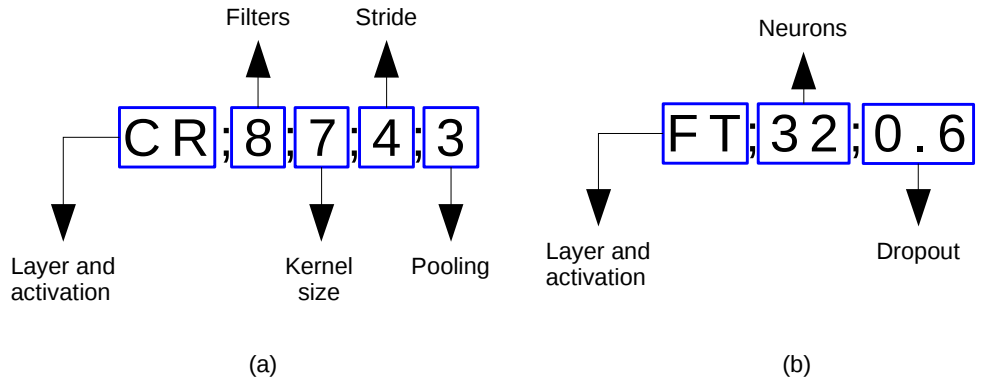


Figure 5.8: Layer descriptor examples. In (a), a convolutional layer, activated by ReLU, with 8 filters, a 7x7 kernel, stride of 4 and pooling of 3. In (b), a fully connected layer, activated by TanH, with 32 neurons and a dropout rate of 0.6.

5.5.1 Random search

To evaluate random search performance, a set of 4000 distinct solutions was randomly generated, following the search space restrictions presented in Table 5.4. These same restrictions were used by the GA-based methods.

In order to preserve the random nature of the search method, no guidance was set for model generation. When invalid models are created (for example, because of spatial reduction), they're replaced with valid ones. Each model was then trained for

Table 5.4: Search space restrictions.

Description	Value(s)
# Convolutional layers	0 - 3
Convolutional filters	{2, 4, 8, 16, 32}
Kernel size	{3, 5, 7, 9, 11}
Stride	{1, 2, 3, 4}
Pooling	{2, 3, 4}
Activation functions	{Linear, Sigmoid, TanH, ReLU}
# Fully connected layers	0 - 2
Neurons	{8, 16, 32, 64}
Dropout	{0.0, 0.2, 0.4, 0.6}
Activation functions	{Linear, Sigmoid, TanH, ReLU}

100 epochs, using the standard Adam optimization method, with parameters $\alpha = 0.001$, $\beta 1 = 0.9$, $\beta 2 = 0.999$ and $\varepsilon = 10^{-8}$ [Kingma and Ba 2014]. Batch sizes of 64 for the MNIST dataset and 50 for the CIFAR-10 dataset were used. To prevent resource waste, an early stopping of 10 epochs was set up, so that training is interrupted when no improvement in the validation accuracy is observed for 10 consecutive epochs.

5.5.2 Grid search

The objective of grid search is the exhaustive exploration of a solution space. However, this is impractical in deep learning topology optimization, given the high number of hyper-parameters to be configured. Therefore, to provide a fair comparison setup, a more restrictive subset of the search space, presented in Table 5.4, was chosen. This subset, presented in Table 5.5, contains a total of 3654 models, resulting from all combinations of its hyper-parameter values. In grid search evaluation, discarded invalid solutions were not replaced, to preserve strict exploration of the reduced search space.

Table 5.5: Search space restrictions for grid search.

Description	Value(s)
# Convolutional layers	0 - 3
Convolutional filters	{16, 32}
Kernel size	{3, 5}
Stride	1
Pooling	2
Activation function	ReLU
# Fully connected layers	0 - 2
Neurons	{32, 64}
Dropout	0.4
Activation functions	{Sigmoid, TanH, ReLU}

5.5.3 Genetic algorithm

The GA-based search was conducted using the parameters presented in Table 5.6. By defining a population size of 200 and 20 generations, a maximum of 4000 solutions is to be evaluated. Crossover and mutation probabilities were set to 70% and 20% respectively. An elitism parameter was configured so that the 10% fittest solutions are automatically propagated to the next generation. The remaining solutions will be obtained through recombination of the current population. The standard roulette wheel was used as the selection method, so that each solution has a selection probability proportional to its fitness.

Table 5.6: GA parameters.

Description	Value(s)
Population size	200
Generations	20
Crossover probability	0.7
Mutation probability	0.2
Elitism	20
Selection method	Roulette

5.5.3.1 Chromosome

Each solution $s_i, i = 1, \ldots, 200$ is represented by two stacks of layer descriptors, one for convolutional layers and the other for fully connected layers. Therefore, each chromosome is composed of two vectors of strings, each string being the descriptor of a layer as presented in Figure 5.8. Convolutional layers descriptors contain the activation function, number of filters, kernel size, stride and pooling size. Fully connected layers descriptors contain the number of neurons and the dropout probability. These solutions compose the initial population, containing therefore 200 randomly created models.

5.5.3.2 Evaluation

The fitness function, used to evaluate a given solution, is represented by the validation accuracy of the solution in the corresponding classification problem. Through indirect encoding, the solution's layer descriptors are used to build a deep learning model, which is trained using the same procedure described in Section 5.5.1. The best accuracy of the model when classifying test samples (samples which are not present in training), calculated using Equation 5.1, is then used as the solution fitness. Once a model is trained, it is stored in a local database, to prevent it from being re-evaluated if it is present in further generations.

5.5.3.3 Selection

Solutions are selected for recombination based on their fitness (i.e. validation performance) by using the roulette wheel selection method, as described in Chapter 2. Each solution has a probability of being selected which is directly proportional to its contribution to the sum of all individual's fitness, providing better solutions with greater chances of delivering their genetic material to the next generation of solutions.

5.5.3.4 Crossover

Crossover is performed by swapping layers between 2 parent solutions, according to the randomly chosen cut-points. Since each solution is composed by two stacks of layers (convolutional and fully connected), the recombination is conducted independently for each stack, using the standard single-point crossover [Mitchell 1998]. For each

layer stack, a random floating-point $p \in [0, 1]$ is generated, and crossover is conducted if $p <= 0.7$.

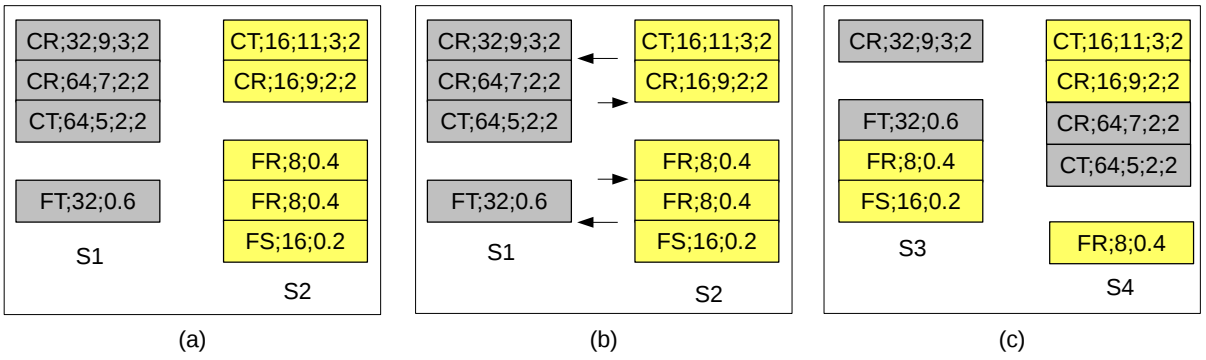


Figure 5.9: Crossover example - solutions S3 and S4 are produced from crossover of parents S1 and S2.

In Figure 5.9 (a), two parent solutions (S1 and S2) are presented. Solution S1 has 3 convolutional and 1 fully connected layer, while S2 has 2 convolutional and 3 fully connected layers. The cut-points of both stacks of each parent are indicated using arrows in Figure 5.9 (b). Finally, in Figure 5.9 (c), the two children solutions resulting from the crossover operation are presented: S3 has one convolutional layer and 3 fully connected layers, while S4 presents 4 convolutional and 1 fully connected layer.

5.5.3.5 Mutation

The mutation operation consists on adding or removing layers at random positions in the solution's chromosome. As with crossover, mutation is conducted independently, for convolutional and fully connected layers. For each layer stack, a random floating-point $p \in [0, 1]$ is generated. If $p \leq 0.2$, a second random floating-point $p_{add} \in [0, 1]$ will be generated, indicating the type of mutation. If $p_{add} \leq 0.5$ a random layer will be added at a random position, in the corresponding layer stack. Otherwise, one of the existing layers will be removed from the corresponding layer stack.

In Figure 5.10(a), the two solutions previously obtained after crossover are presented. The arrows in Figure 5.10(b) indicate mutation points: for solution S3, a new random layer will be added after the arrow. For S4, the layer pointed by the arrow will be removed. The two solutions presented in Figure 5.10(c) are the resulting solutions of these operations.

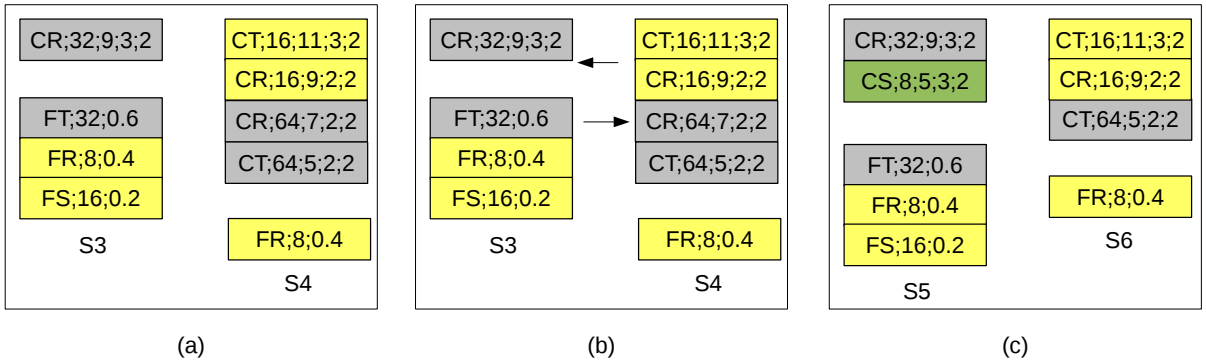


Figure 5.10: Mutation example - solutions S5 and S6 are produced after mutation of S3 and S4.

After crossover and mutation, valid produced solutions are evaluated and added to the population. Invalid solutions are discarded, and new recombinations take place until the population is complete with 200 individuals.

5.5.4 Genetic algorithm with fitness prediction

Many of the solutions evaluated in the previous methods present a low test accuracy after deep learning model training. This is an undesirable effect, since computational resources were allocated to train a model which will be further discarded. For this reason, a modified version of the GA-based search was experimented. In this version, new generated solutions are subject to a performance prediction step. The predicted performance will then be used to determine if this solution is worth evaluation. To achieve this, an ANN-based predictor was setup, using knowledge gained from previous evaluations to estimate new models performance.

The fitness predictor was setup with a fixed topology of 3 fully connected layers, with 64 neurons each, and a dropout rate of 0.4. The inputs of the predictor are formatted as a vector of integers containing the descriptors of the model layers. Each convolutional layer is represented by a 9 element vector, containing the number of filters, kernel size, stride, pooling, normalization and the categorized activation function. Each fully connected layer is represented by a 6 element vector, containing the number of neurons, categorized activation function and the dropout rate. The final input vector contains 39 integers, composed of 3 convolutional descriptors and 2 fully connected descriptors. Empty layers are represented by a series of zeros in the corresponding

positions. Figure 5.11 presents examples of the predictor inputs.

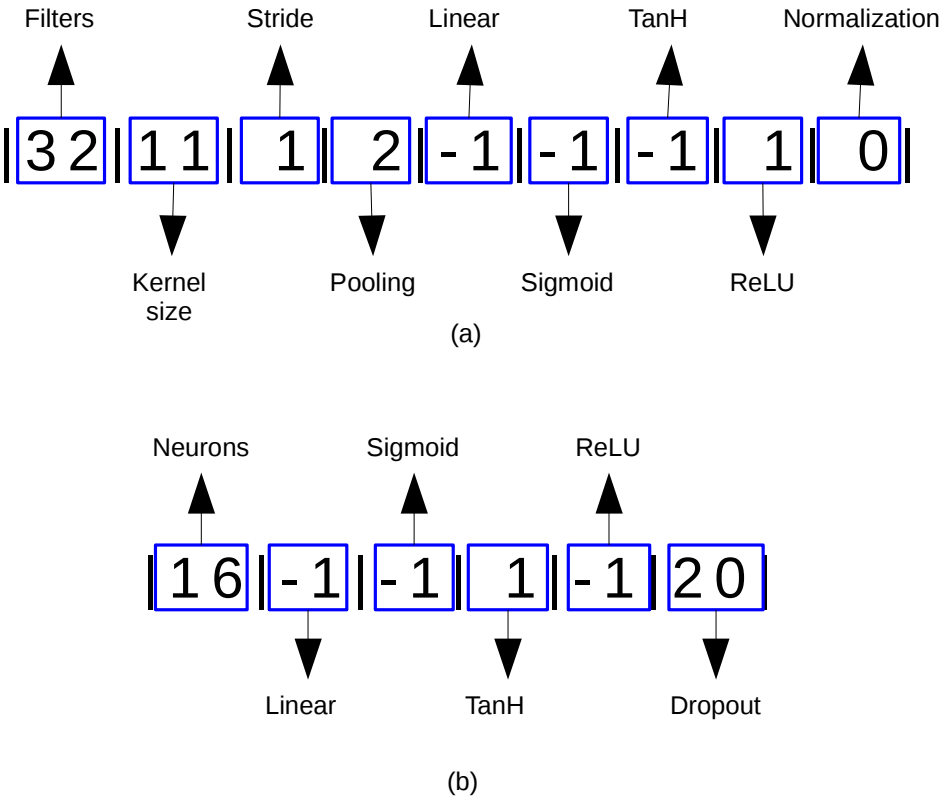


Figure 5.11: Predictor input examples. In (a), the descriptor CR;32;11;1;2 is represented. In (b), the descriptor FT;16;0.2 is represented.

The fitness predictor was trained for 100 epochs, with an early stopping of 10 epochs (validation set), using the same optimizer as the MNIST and CIFAR-10 classifiers. The available training data (set of models previously trained and the corresponding fitness) is split into training (80% of the available data) and validation (20% of the available data). When a solution is evaluated, it feeds the ANN predictor, which is re-trained with the new pattern.

The decision on whether to evaluate a model or not is made upon the comparison of the predicted fitness with the already evaluated models. To this end, the Pareto efficiency was used, considering two dimensions: the solution fitness (performance of the DNN model) and the number of weights (complexity of the model). Dominant solutions (i.e. solutions that are superior to others in all attributes) have a 90% probability of being evaluated, while dominated solutions have a 10% probability of being evaluated. These probabilities were defined empirically, in order to provide promising solutions with more computational resources, when compared to potentially bad solutions. The

complete algorithm used in this search method is described in the next paragraphs.

In Figure 5.12, the main tasks for generating the initial population are presented.

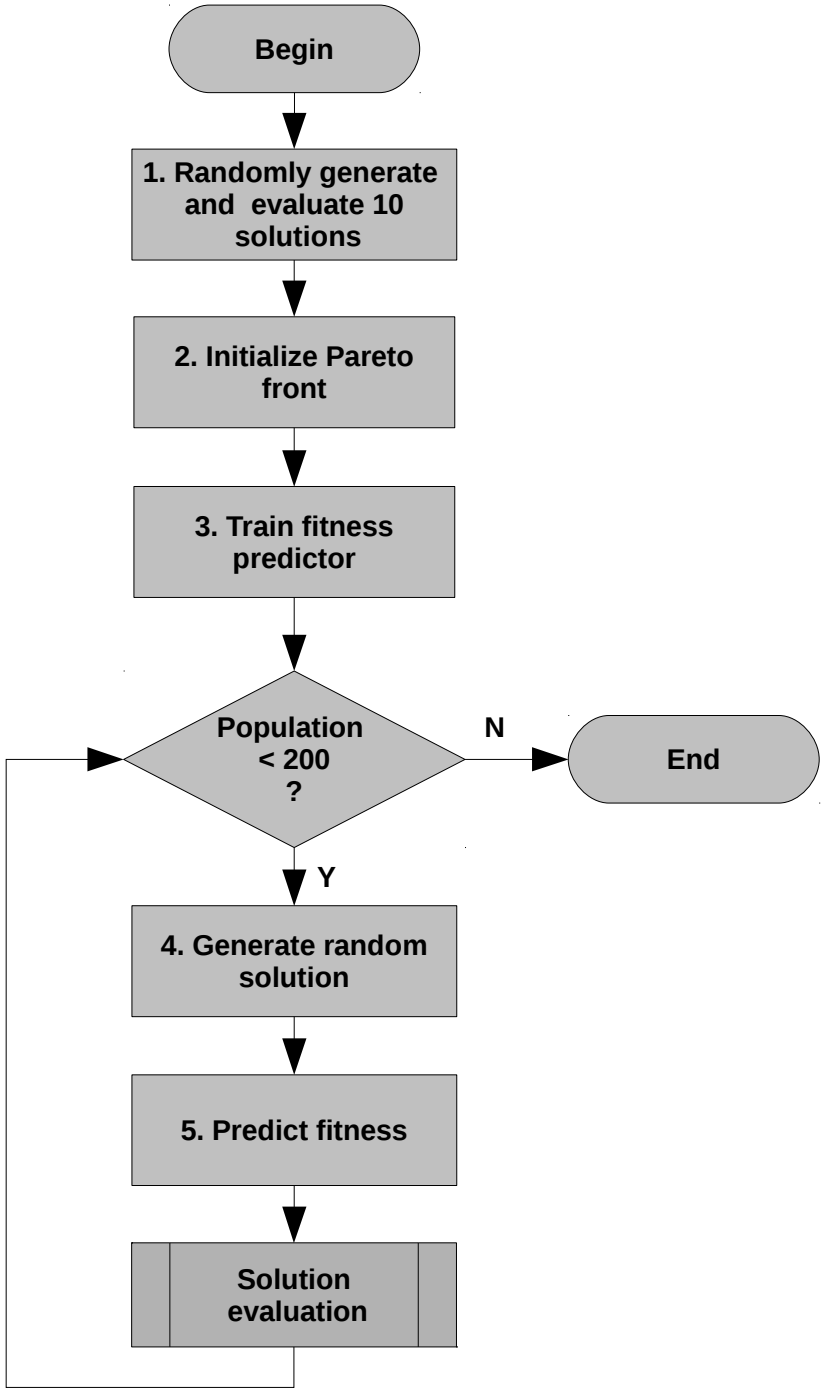


Figure 5.12: Initial population generation.

1. Initially, 10 solutions are randomly generated and evaluated.
2. The Pareto front is created, containing the dominant solutions among these 10 solutions.

3. The fitness predictor is trained, using the 10 solutions as input patterns. 8 solutions are randomly chosen to be used as training patterns, while the 2 remaining solutions are used as validation patterns.

4. While the number of solutions in the population is less than the desired population size (200), new solutions are randomly generated.

5. The predictor described in task 3 is used to predict the fitness of the new generated solution. This predicted fitness, together with the number of weights of the solution, will be used to determine if its network model will be trained or not. The procedure for solution evaluation is presented in Figure 5.13.

Figure 5.13 presents the main steps for solution evaluation.

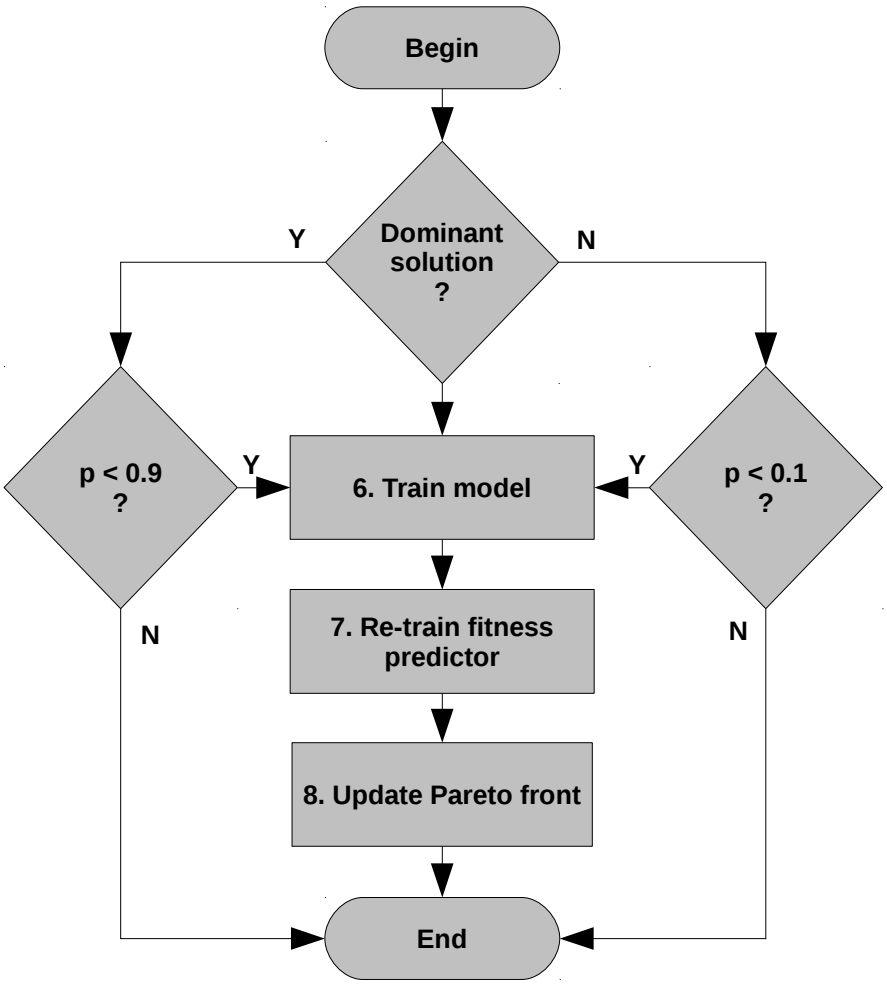


Figure 5.13: Solution evaluation.

6. If the solution (with predicted fitness) belongs to the Pareto front (dominant solution), it has a 90% probability of being trained. Otherwise (dominated solution), this probability decreases to 10%. This step has the objective of providing more resources to potentially good solutions, while preventing resource wasting by bad solutions. If a random generated number (p) is below the respective threshold, the solution's model is trained.

7. After model training, the fitness predictor is updated, adding the new solution to the input patterns and being re-trained with the new data. As in previous training, 80% of the available patterns are used for training, while the remaining patterns are used for test.

8. Finally, the Pareto front is updated with the new solution.

After the initial population is generated, it is evolved through the desired number of generations. In the results reported in this work, the population is evolved for 20 generations. The steps for population evolution are presented in Figure 5.14.

9. If the number of generations is not achieved, a new generation is initialized with an elite of the 20 best solutions (10%) of the current population.

10. Elite solutions which weren't previously evaluated (predicted solutions) have their models trained.

11. While the number of solutions in the current generation is less than the population size (200), two solutions are selected from the current population, using the roulette selection algorithm.

12. The crossover operator (as described in Section 5.5.3.4) is applied, with a probability of 70%.

13. The mutation operator (as described in Section 5.5.3.5) is applied, with a probability of 20%.

14. When the number of desired solutions is achieved, the population is updated with the solutions in the current generation.

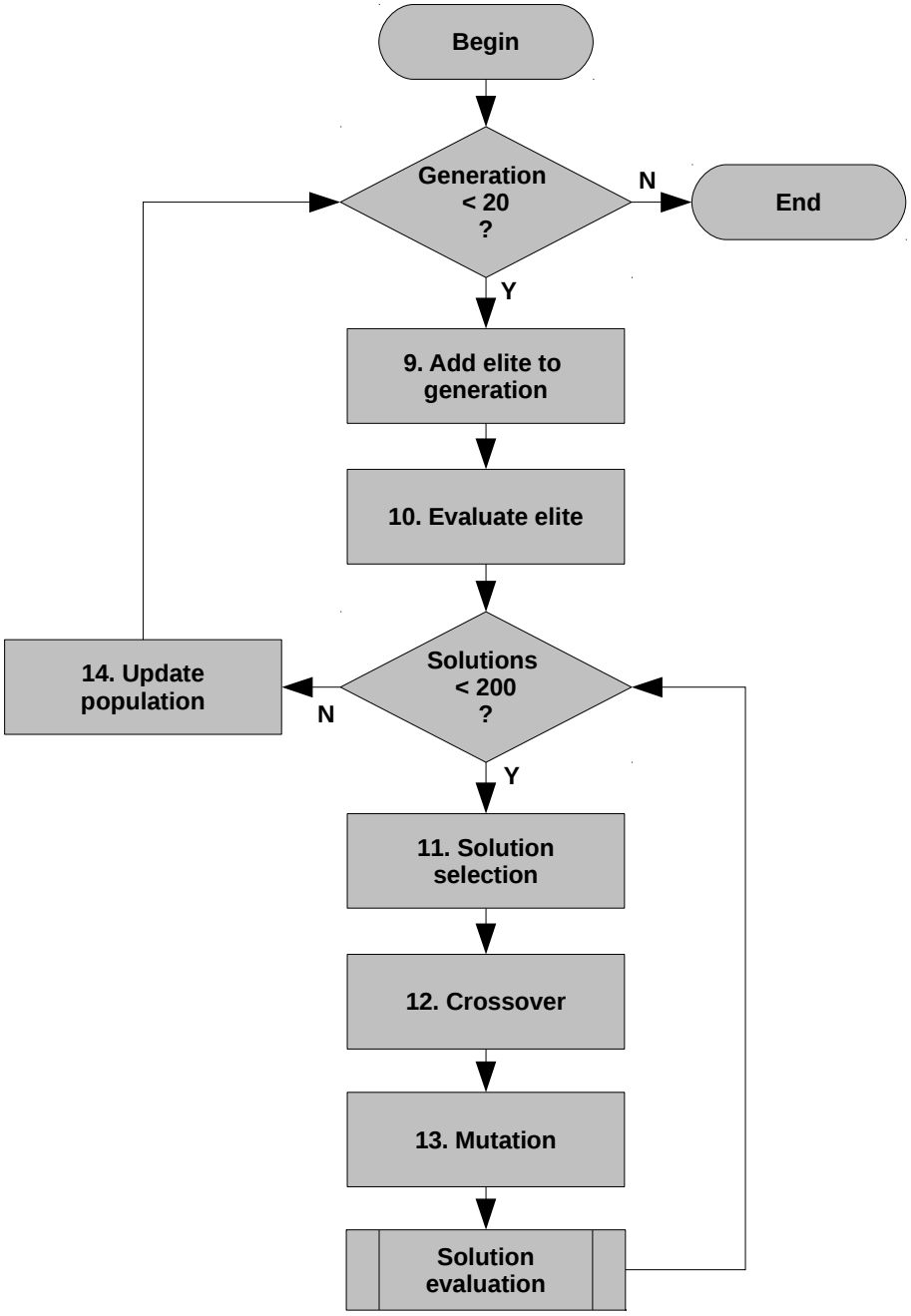


Figure 5.14: Population evolution.

5.6 Final considerations

This chapter presented implementation details of the main contributions of this work, describing the architecture model, the design procedure of developed deep visual trackers and the experiments conducted to evaluate the application of the evolutionary algorithm to network topology selection. The presented architecture model makes it possible to integrate components from heterogeneous providers in a distributed AR

application. The presented topology search algorithms can be used to assist the design of deep neural networks in visual tracking, and also be adapted to many other applications, given the genericity of designed representation and operators. In the next chapter, results obtained with the conducted experiments will be presented and discussed.

Chapter 6

Results

This chapter presents the results obtained with the conducted experiments described previously in this thesis. The interpretation of these results as well as their contribution are also discussed.

6.1 Software architecture for distributed AR

Table 6.1 presents the average frame rate, on the client's endpoint, for both proofs of concept of the architecture model presented in this work. The nine scenarios (using the fixed/rotating marker videos) were investigated, but no significant differences were observed in experimental results, except for a slight decrease in the WebRTC/JSARToolKit prototype, for 3 and 5 z-rotating markers.

Figure 6.1 presents CPU utilization for both prototypes, in each of the 9 experiments (media server side). As WebRTC is executed on user's browser, the collected values represent the browser resource utilization.

As more markers are added to the testing video, marker detection and pose estimation routines lead to an increase on CPU utilization, for both prototypes. As the number of markers increases, a small increase in memory consumption was also observed, except for the z-rotating dataset (Figure 6.2). In this particular experiment, a decrease in memory consumption was observed in the WebRTC/JSARToolKit prototype, from 3 to 5 markers. Also, memory consumption also decreased in the OpenCV/libVLC prototype, from 1 to 3 markers.

Table 6.2 presents the average and standard deviation of processing and transmis-

Table 6.1: Frames per second on clients' endpoint. *S* refers to static markers, while *Ra* refers to rotating markers around axis *a*. 1, 3 and 5 are the number of markers contained in each per frame.

	OpenCV/libVLC		**WebRTC/JSARToolKit**	
Video label	**AVG(FPS)**	**SD**	**AVG(FPS)**	**SD**
S1	29.94	0.90	24.83	2.11
S3	29.95	0.67	24.67	2.22
S5	29.94	0.71	24.52	2.12
RY1	29.95	0.78	24.57	2.11
RY3	29.95	0.56	24.43	2.16
RY5	29.95	1.44	24.23	2.29
RZ1	29.95	0.78	24.62	1.93
RZ3	29.96	1.26	22.31	3.47
RZ5	29.96	0.59	22.00	4.09

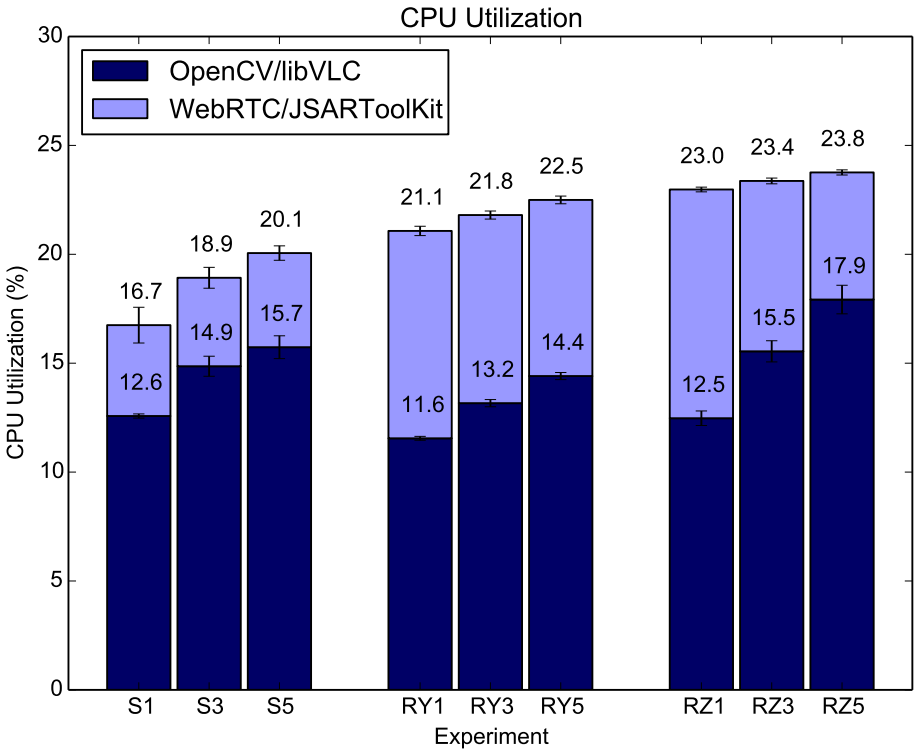


Figure 6.1: CPU utilization for OpenCV/libVLC and WebRTC/JSARToolKit prototypes.

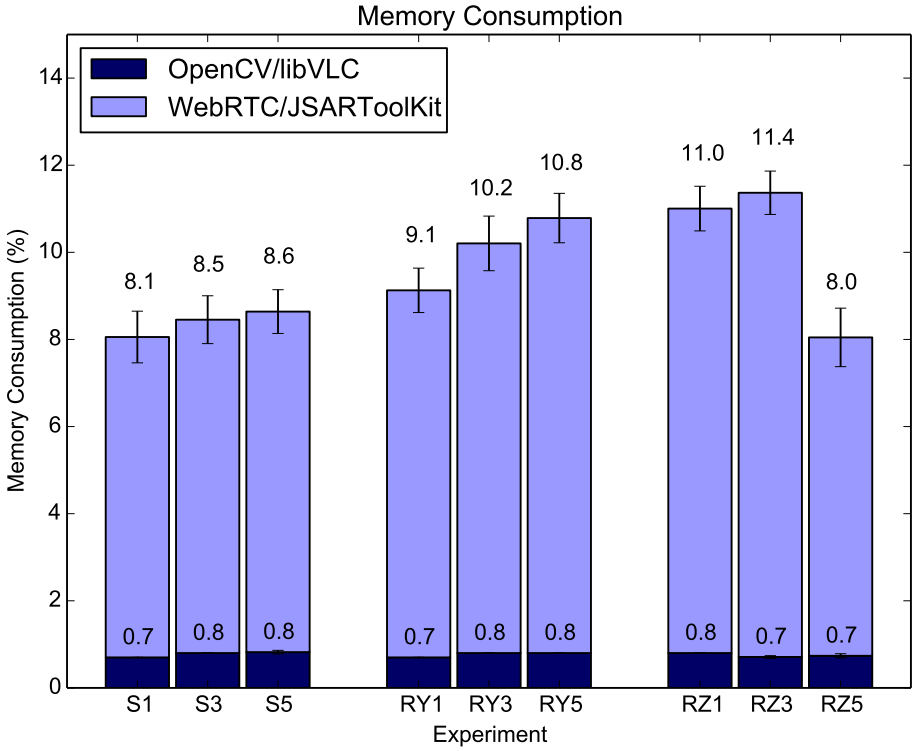


Figure 6.2: Memory consumption for OpenCV/libVLC and WebRTC/JSARToolKit prototypes. Data collected at the media server side, running on an 8 GB desktop.

sion delays on the client's endpoint.

Table 6.2: Processing / transmission delays.

	Δt_p (ms)		Δt_t (ms)	
Prototype	**AVG**	**SD**	**AVG**	**SD**
OpenCV/libVLC	4.62	2.23	759.29	7.43
WebRTC/JSARTK	11.79	3.55	585.31	111.97

The OpenCV/libVLC prototype presented better performance for AR processing, when compared to the browser based implementation. However, observed end-to-end transmission delay was shorter in the latter. Since the WebRTC/JSARToolKit implementation is executed inside the browser context - as opposed to the standalone OpenCV/libVLC - it is more susceptible to the effects of multi-processing and multi-threading of the browsers' tasks, which leads, among other effects, to greater variation of the collected metrics.

Figures 6.3 and 6.4 present SSIM index and PSNR for the collected frames, for both

OpenCV/libVLC and WebRTC/JSARToolKit prototypes. All collected frames presented SSIM indexes above 0.88, with the majority of frames above 0.92. For the PSNR metric, all collected frames presented values above 36 dB, with the majority of frames between 38 and 42 dB. While the desktop based prototype presented better results for the SSIM index, higher PSNR values were observed for the browser based solution. The observed difference is explained by the distinct application of each metric: while SSIM is related to the perception of image quality, PSNR is calculated using the mean squared error of target and reference images.

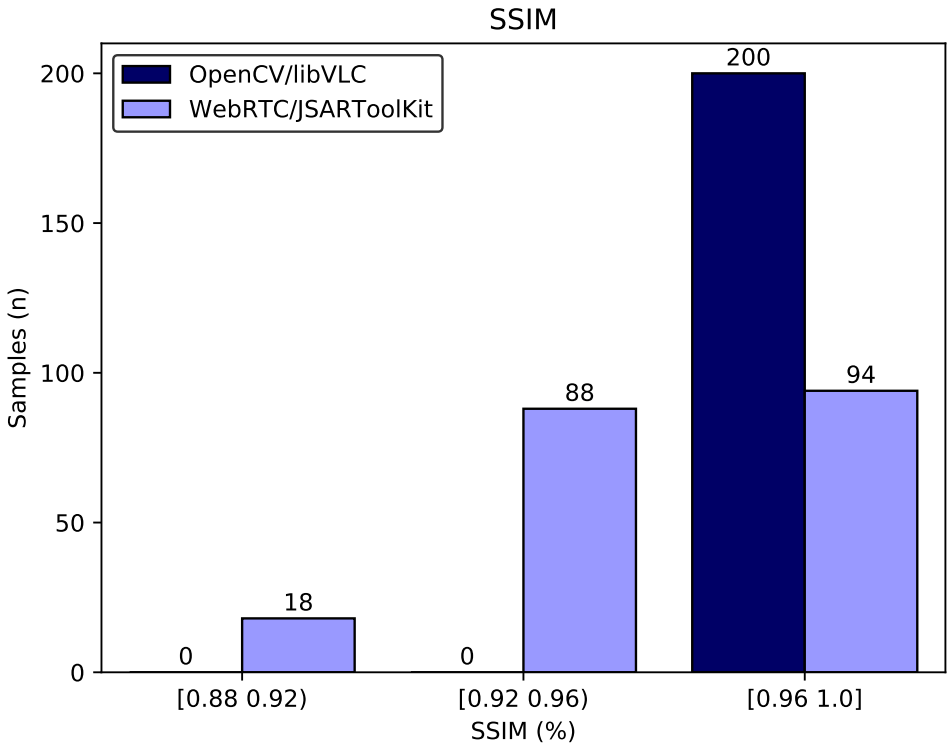


Figure 6.3: SSIM for OpenCV/libVLC and WebRTC/JSARToolKit prototypes.

Presented experiments made it possible to evaluate both implementations of the proposed architecture model. The OpenCV/libVLC implementation achieved a frame rate above 29 FPS for all reference videos, which is a satisfactory metric given the 30 FPS reference value from the literature [Chouiten, Didier and Mallem 2012]. Results on image quality were also promising: all frames transmitted during the corresponding test presented an SSIM index above 0.96; the same frames presented a PSNR value above 36 dB, with 75 % of the frames over 38 dB. As reported in the literature, these values correspond to “good” (above 31 dB) and “excellent” (above 37 dB) quality, in

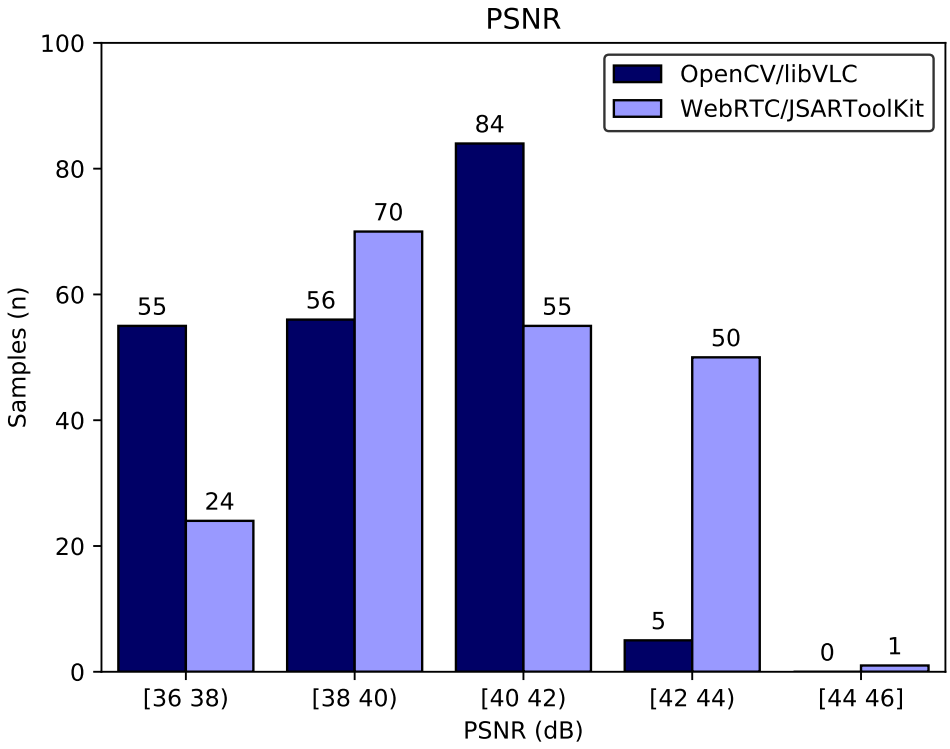


Figure 6.4: PSNR for OpenCV/libVLC and WebRTC/JSARToolKit prototypes.

the subjective Mean Opinion Score (MOS) scale, commonly used in video transmission quality evaluation [Klaue, Rathke and Wolisz 2003, Migliorini, Mingozzi and Vallati 2011].

Data collected with the presented delay model indicate an average processing delay of 4.62 ms for the OpenCV/libVLC prototype and 11.79 ms for the WebRTC/JSARToolIKit implementation. Observed average transmission delay was of 759.29 ms for the first and 585.31 ms for the second implementation. This value is still above the desired value of 100 ms, adopted in the literature as a reference value for real-time applications [Chouiten, Didier and Mallem 2012]. Therefore, in order to match this requirement, improvements should be made in the streaming layers of the architecture. In this sense, tests involving fine tuning of the current streaming components, as well as other streaming alternatives will be conducted.

6.2 Deep visual tracking for AR

In this section, the results of the experiments conducted to evaluate the performance of deep visual trackers in AR tracking are presented. AR marker tracking can be split in two main steps, namely marker detection (presence or absence of marker in frame) and marker localization (key point coordinates localization). The metrics used to evaluate each of these steps are presented in the next paragraphs.

6.2.1 Marker detection

To evaluate the performance of the neural networks in AR marker detection, a dataset with 1527 sample images were used for training. 300 images from the datasets described in Section 5.4 were randomly selected for network testing. Two detectors based on the ARToolKit library [1] and the OpenCV framework [Baggio 2012] were used to establish the reference values (whether an AR marker is present or not in each image). Using these reference values, DNN accuracy in marker detection (ACC) were calculated on the test images, using equations 6.1, 6.2 and 6.3.

$$ACC = \frac{n_c}{n_c + n_w} \tag{6.1}$$

$$n_c = t_p + t_n \tag{6.2}$$

$$n_w = f_p + f_n \tag{6.3}$$

where n_c is the number of correctly detected markers, n_w the number of detection failures, t_p the number of true positives, t_n the number of true negatives, f_p the number of false positives and f_n the number of false negatives. Validation set accuracy of the best configuration from each model during training can be observed id Figure 6.5. The minimum, average and maximum value for testing accuracy, as well as the number of evaluated configurations for each model are summarized in Table 6.3.

The results obtained in this experiment indicate a small increase in accuracy for the models presenting convolutional layers, when compared to the models using only

[1] https://artoolkit.github.io/portal/index.html

fully connected layers. The best test accuracy (98.46%) was obtained with configuration CR(16,17,4) - MP(2,2) - C(4,11,4). This configuration was used further in the experiments presented in Section 6.2.3.

6.2.2 Marker localization

In a second experiment, the performance of neural networks in AR marker localization was evaluated. For network training, the images of the original dataset containing AR markers were used, resulting in 600 training imagens. For network testing, 150 images containing AR markers were randomly sampled from the datasets described in Section 5.4. The ground truth values - corresponding to the coordinates of 4 marker vertices - were obtained using the same detectors described in the previous experiment. The performance of each configuration was measured using the root mean squared error (RMSE) - commonly used in experiments approaching keypoint detection [Berretti et al. 2011, Zhang et al. 2014] - as presented by Equation 6.4.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} ((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2)}{N}} \tag{6.4}$$

where (x, y) are the ground truth coordinates, $(\hat{x}, \hat{y})$ the predicted coordinates and N the number of key points.

Figure 6.6 presents the training error, for the best configuration of each experimented model. Table 6.3 presents the obtained results for the testing samples, including minimum, average and maximum values of the RMSE.

The results obtained with this experiment indicate a better performance of models containing fully connected layers when compared to the model containing only convolutional layers for the training data. However, for test data, models containing only convolutional layers presented better performance (as indicated by Table 6.3). This behavior can be justified by the possible overfit of models containing fully connected layers, given the small number of patterns used in training (600 patterns). The use of overfitting prevention techniques - such as dropout - can contribute to reduce this effect, and was explored in the experiments described in Section 6.4. The configuration CR(12,13,4) - C(8,11,4) were selected for further analysis, since it presented the smallest error for the test samples.

Table 6.3: Marker detection and localization metrics after 1000 training epochs, with early stopping of 50 epochs. Statistics refer to the ensemble of models evaluated in each layer type, considering the best test loss of each model in the target dataset.

Dataset	**Layer types**	**Best model**	n	**Metric**		
				Accuracy (%)		
				MIN	**AVG (STD)**	**MAX**
AR (Detection)	FC	FC(8) - FCR(4) - FCR(8)	64	58.15	92.45 (7.10)	97.85
	CONV	CR(16,17,4) - MP(2,2) - C(4,11,4)	131	41.85	94.37 (10.54)	98.46
	CONV + FC	CR(12,11,4) - MP(2,2) - LRN(3,1,5) - FC(8)	85	41.85	94.69 (8.34)	98.15
				RMSE (px)		
				MIN	**AVG (STD)**	**MAX**
AR (Localization)	FC	FC(8) - FC(8)	24	1.27	4.40 (3.65)	15.08
	CONV	CR(12,13,4) - C(8,11,4)	20	0.66	1.61 (0.82)	3.33
	CONV + FC	CR(20,9,4) - FC(16)	22	0.79	1.37 (0.78)	3.56

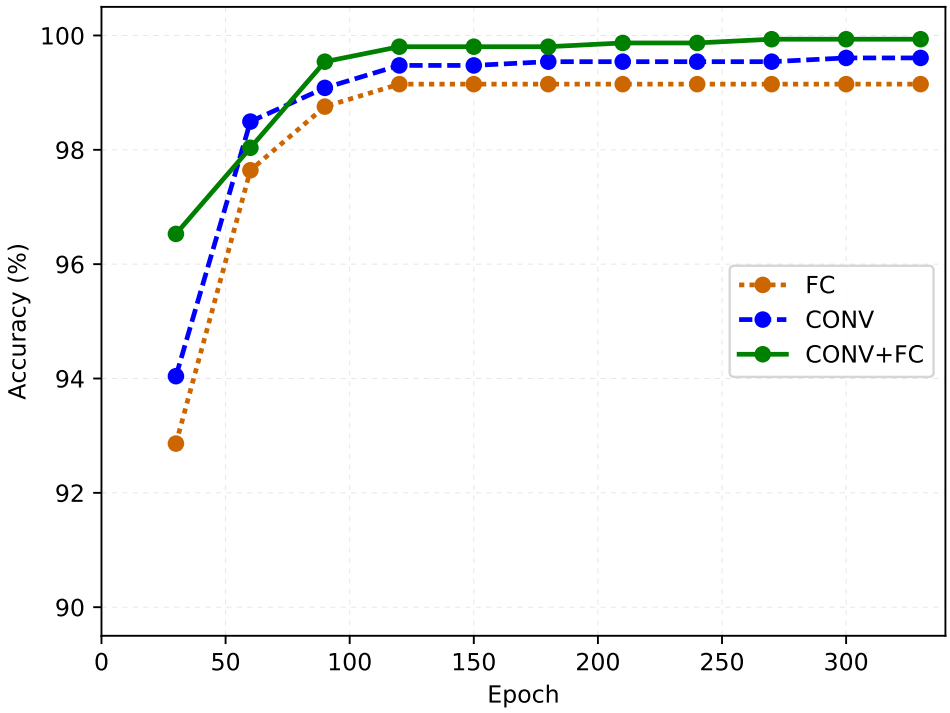


Figure 6.5: Marker detection: accuracy across training epochs (best configuration for each model).

6.2.3 Marker tracking in streamed videos

The networks presented in the previous sections were used for detection and localization of AR markers in the datasets containing videos streamed in different conditions (as described in Section 5.4). The performance of each configuration in marker detection was measured using the false negative rate (FNR), as presented by Equation 6.5 [AlBasiouny, Sarhan and Medhat 2015].

$$FNR = \frac{f_n}{f_n + t_p} \tag{6.5}$$

where f_n is the number of non detected markers and t_p is the number of markers correctly detected. The obtained results, as well as a comparison with the reference detectors are presented by Figure 6.7. The localization errors of each configuration were measured using the maximum RMSE for each dataset, and are presented in Figure 6.8.

Deep visual trackers have outperformed the reference implementations in marker

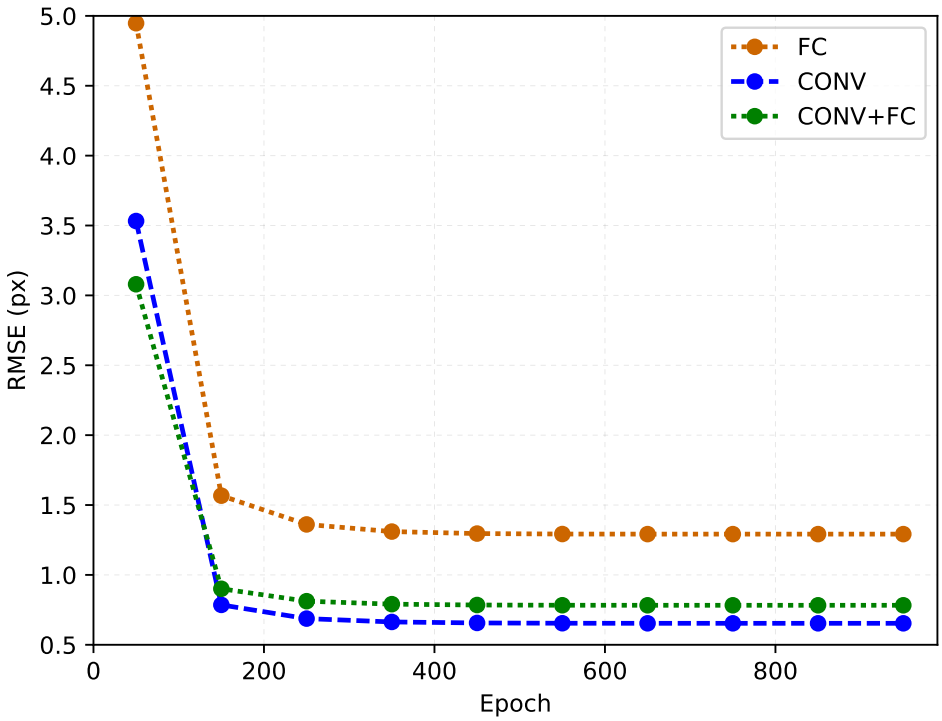


Figure 6.6: Marker localization: error across training epochs (best configuration for each model).

detection for the datasets produced using WebRTC and RTSP. These results indicate a better robustness of these trackers, when applied to images with streaming quality loss. For marker localization, deep visual trackers presented greater values for the error (although close) when compared to the ARToolKit implementation. Both outperformed the OpenCV implementation. The values of these metrics, in addition to the processing delay observed for marker detection (7 ms/frame) and localization (8 ms/frame), confirm the viability of deep learning techniques applied to marker tracking in Augmented Reality systems, for remotely captured and streamed videos. Figure 6.9 presents frame samples and their corresponding localization errors, for each of the trackers evaluated in this work.

6.3 Evolutionary selection of DNN topologies

After testing the random, grid and GA search methods described in Chapter 5, it was possible to identify some contributions of the evolutionary-based techniques presented

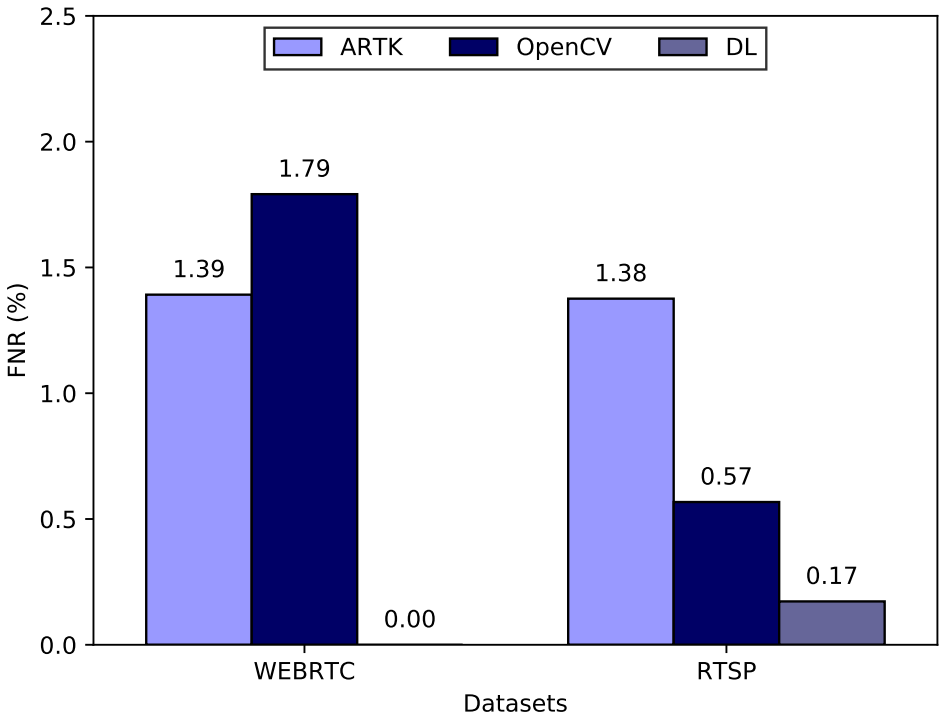


Figure 6.7: Detection errors for streamed videos.

in this work. These contributions will be detailed in this section.

Figures 6.10 and 6.11 present the evaluated solutions for the 4 search methods (random, grid, GA and GA + predictor) highlighting the Pareto front obtained with each one, respectively with the MNIST and CIFAR-10 datasets. It is possible to notice the concentration of evaluated solutions near the Pareto front, for all experiments. In grid search, the exploration of a restricted solution subspace causes the prevalence of solutions with more weights, diverging from the other approaches which randomly explore the entire solution space. It is also worth mentioning that the Pareto front was determined using equal weights for both attributes (fitness and weights). Therefore, in these results, a small model with low accuracy is equally important as a big model with higher accuracy. To change this behavior, weights can be applied to each attribute, making, for example, accuracy more important than model complexity.

For the CIFAR-10 dataset, although some models achieved 100% accuracy on training samples, all evaluated models presented test accuracy below 60%. As the training and test samples were randomly chosen from the original dataset (preserving equal distribution of samples per class), these results might be explained due to the in-

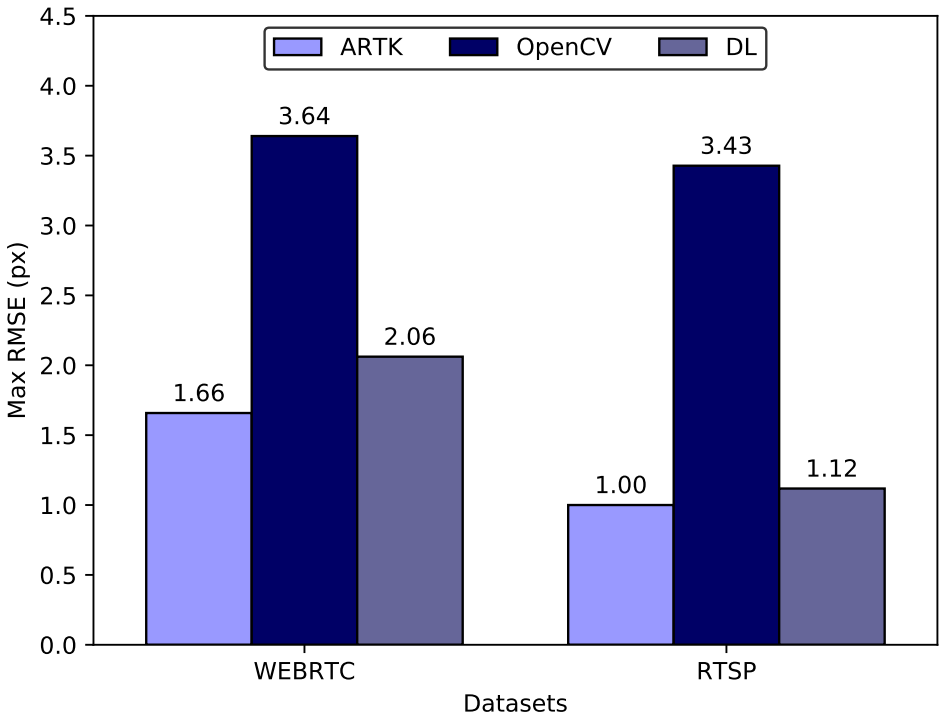
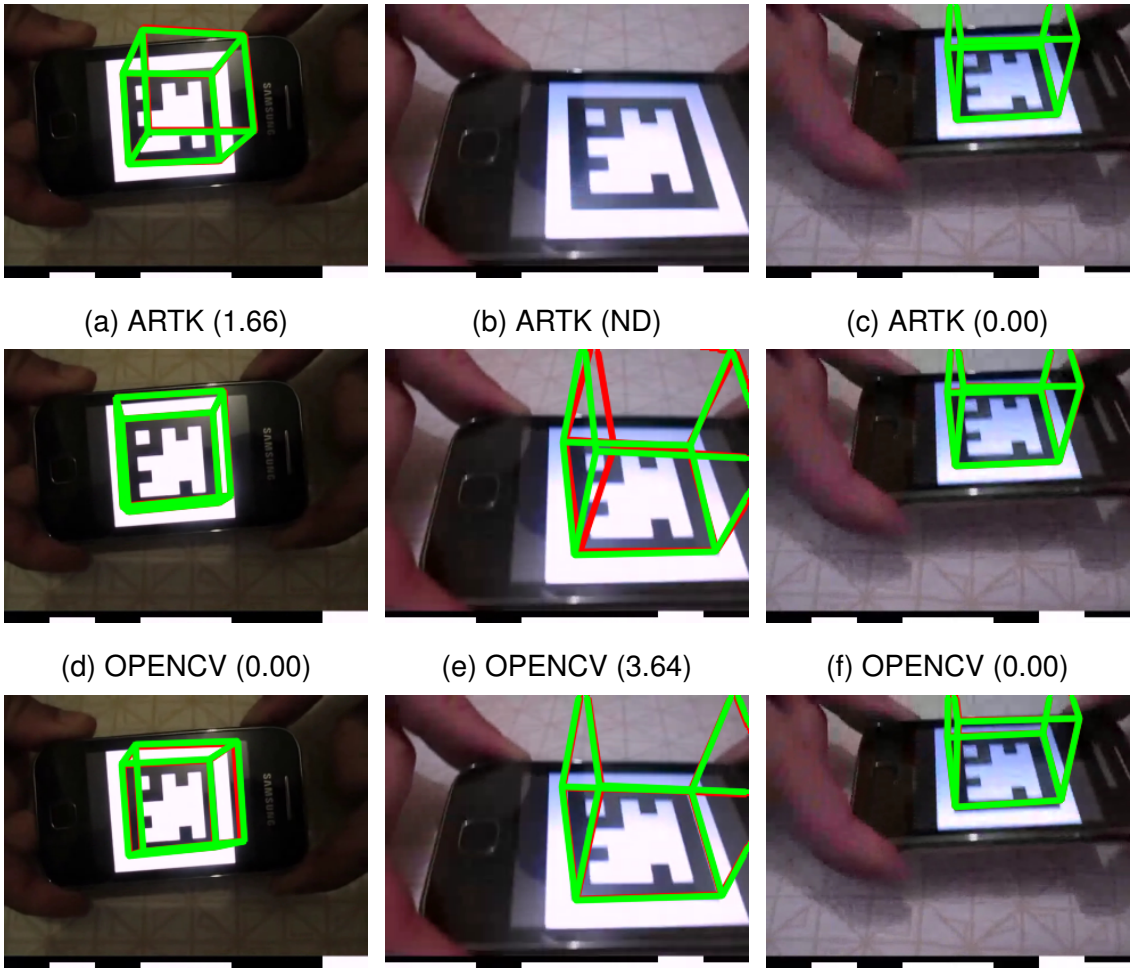


Figure 6.8: Localization errors for streamed videos.

herent complexity of the CIFAR-10 dataset, which would require more training samples to improve testing accuracy. To investigate this hypothesis, the best models were chosen to a second training stage, with more training samples from the original dataset. The results of this evaluation will be presented further in this section.

Figure 6.12 presents the evolution of both the population of solutions and its best individual across all generations after a single run of the GA and GA + Predictor search methods. No significant difference was observed when comparing GA and GA + Predictor results, except for the fact that, with the MNIST dataset in the GA + Predictor experiment, the best solution was found earlier during evolution, which did not occur in the CIFAR-10 dataset. In all cases, the best solutions were kept in the next generations, given the use of elitism. Tables 6.4 and 6.5 summarize the results of the four search methods applied to both datasets.

The GA-based search methods were able to find competitive (and even superior) models compared to the random and grid search methods. However, the number of evaluated models were drastically reduced, as a result of the genetic operators which replicate good solutions in the further generations of the population, causing its con-



(a) ARTK (1.66) (b) ARTK (ND) (c) ARTK (0.00)

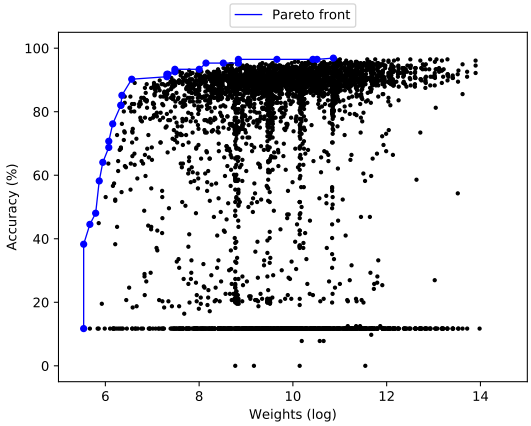
(d) OPENCV (0.00) (e) OPENCV (3.64) (f) OPENCV (0.00)

(g) DL (0.87) (h) DL (1.12) (i) DL (2.06)

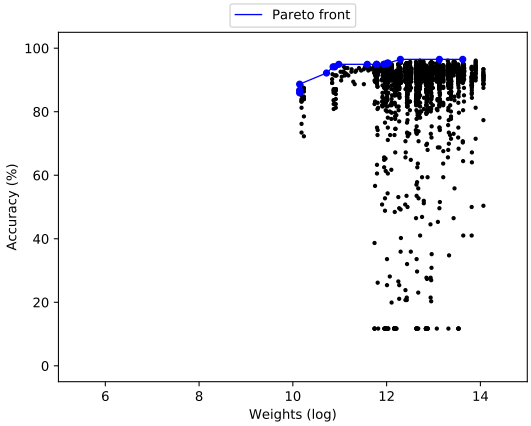
Figure 6.9: Frames with biggest localization errors (in red), for each detector. RMSE is presented between parenthesis. ND indicates a frame with non-detected marker.

vergence to the best evaluated solutions. In grid search, the reduction of the number of evaluated solutions is due to the selected solution space, which contained invalid solutions. This result would be different depending on the search space selection, but this would require prior domain knowledge, differently from the other methods which minimize human intervention.

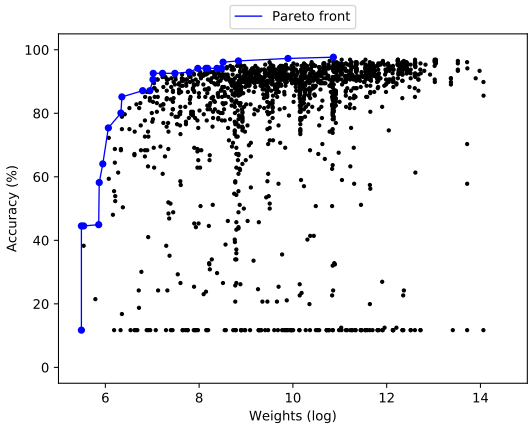
Execution times were measured considering the time spent with optimization (population initialization, selection, genetic operators, etc.), fitness predictor training and solution evaluation (model training). With the MNIST dataset, the GA search method was executed for 15 hours, with 12 minutes spent by the GA. The GA + Predictor method was executed for 14 hours, with 19 minutes spent by the GA and 9 minutes spent with predictor training. With the CIFAR-10 dataset, GA search was executed for



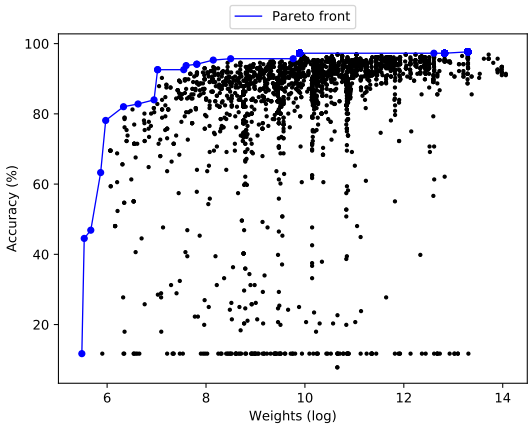
(a) Random search [MNIST]



(b) Grid search [MNIST]



(c) GA [MNIST]

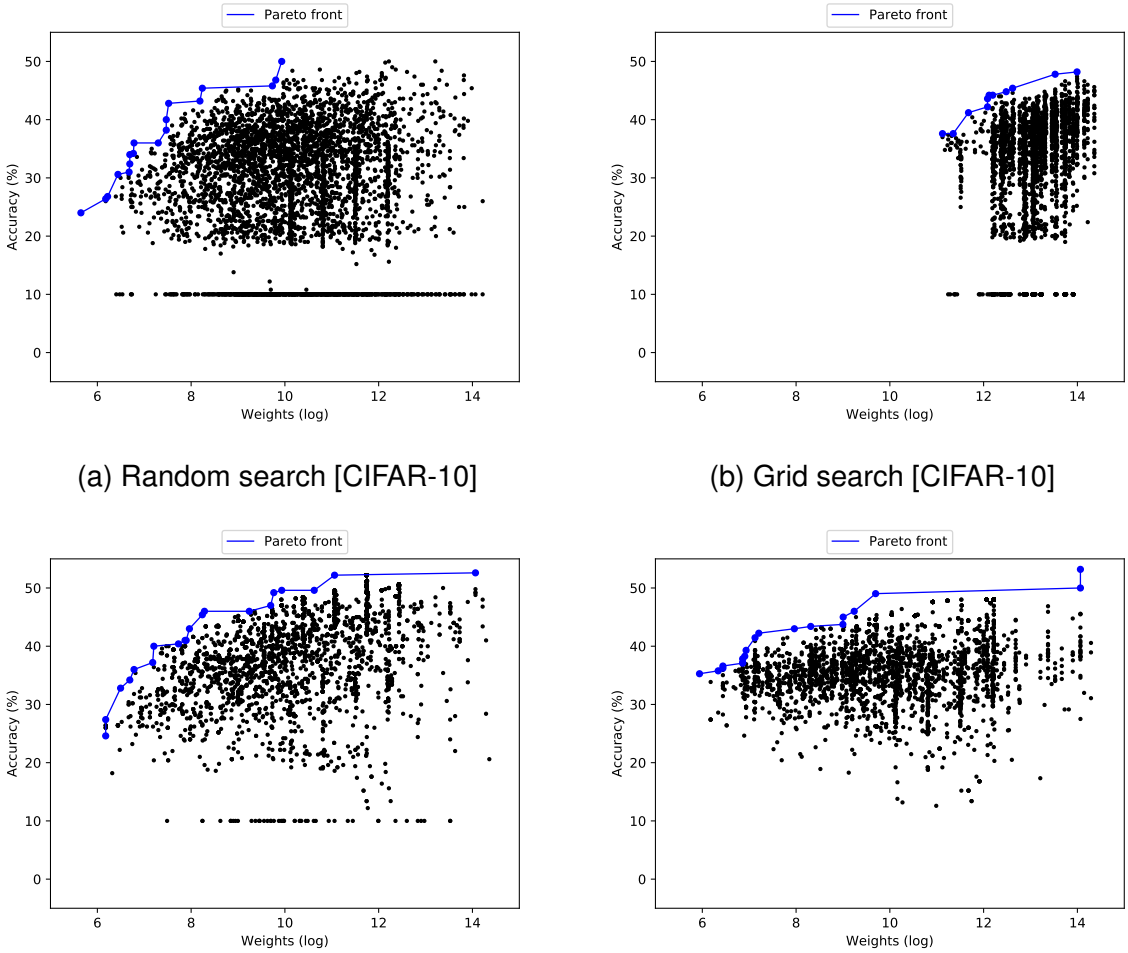


(d) GA + Predictor [MNIST]

Figure 6.10: Pareto front obtained with each search method for the MNIST dataset.

45 hours (8 minutes spent by the the GA), and GA + Predictor search was executed for 24 hours (13 minutes spent by the GA and 8 minutes spent with predictor training). In all these cases, remaining time (total execution time minus optimization and predictor training) was spent with model training. These values show the minimal cost associated with optimization (GA and GA + Predictor search) and fitness prediction (GA + Predictor only), when compared to deep learning model training.

The use of the predictor did not result in significant reduction in the number of evaluated models and execution time for the MNIST dataset, when compared to the standard GA. Also, the best solution found by the GA + Predictor method is far more complex than the one found by the standard GA, which indicates that the prediction efficiency must be improved for this dataset. However, with the CIFAR-10 dataset, a reduction of 43% on the number of evaluated models was observed, in addition to a



(a) Random search [CIFAR-10]

(b) Grid search [CIFAR-10]

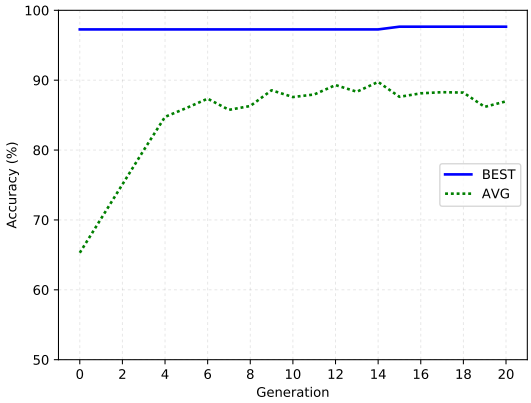
(c) GA [CIFAR-10]

(d) GA + Predictor [CIFAR-10]

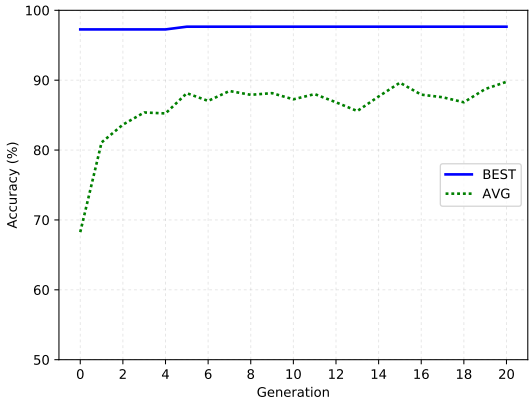
Figure 6.11: Pareto front obtained with each search method for the CIFAR-10 dataset.

significant decrease in execution time. The best solution found by the GA + Predictor method is very similar to the solution found by the standard GA method, except for the activation function and dropout rate in the fully connected layer. These results demonstrate the economy of computational resources observed when prediction accuracy is good enough to discard potentially bad models, and indicate that further investigation on fitness prediction must be conducted to extend this benefit to other datasets.

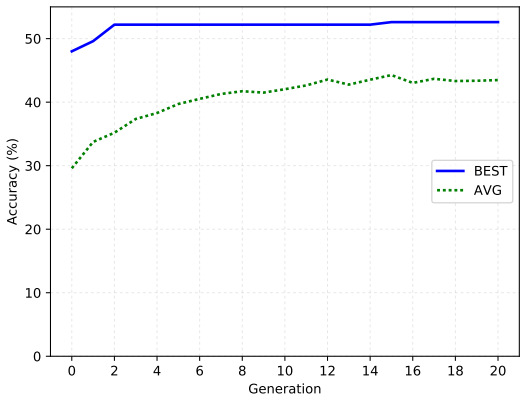
In order to compare the performance of the solutions optimized using the GA with existing models, a larger subset of the MNIST and CIFAR-10 datasets was used. For the MNIST dataset, 10048 training samples and 2048 test samples were randomly selected, while preserving the other configuration parameters. For the CIFAR-10 dataset, 10000 training samples and 2000 test samples were used. As in [Ullah and Petrosino 2016], the optimized models were compared to some of the state-of-the-art



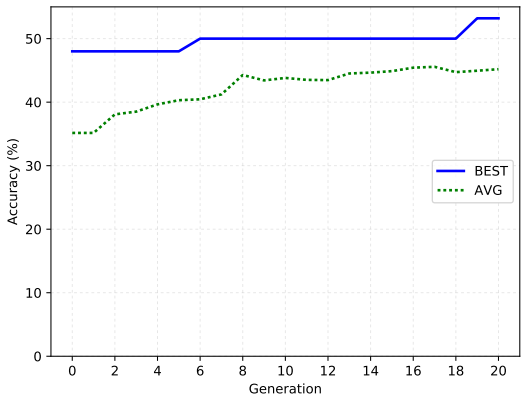
(a) MNIST: GA.



(b) MNIST: GA + Predictor.



(c) CIFAR-10: GA.



(d) CIFAR-10: GA + Predictor.

Figure 6.12: Population evolution for the MNIST and CIFAR-10 datasets.

models, namely LENET [LeCun et al. 1998] and the Caffe Cifar-10 model (C10). In addition, two pyramidal structured models presented in [Ullah and Petrosino 2016] - SPyr_Rev_LENET and SPyr_Rev_C10 - were also evaluated. To provide fair comparisons, all models were trained from scratch using the same training parameters, for a maximum of 100 epochs early stopping when no improvement on the test accuracy was observed for 10 consecutive epochs. Training and test accuracy - measured using Equation 5.1 - as well as the time required to train each model from scratch (Δt) were calculated. The results of this experiment are presented in Tables 6.6 and 6.7.

With the MNIST dataset samples, both standard GA (GANet) and GA + Predictor (GA+PNet) optimized models presented competitive performance when compared to state-of-the-art models, considering both accuracy and training time. In addition, GANet was the fastest model to train, presenting a decrease of 0.4% and 0.5% in

Table 6.4: MNIST summary.

Method	Evaluated solutions	Best solution		
		Model	Acc (%)	Weights
Random search	4000	CTN;32;5;2;3 FL;16;0.6	97	52218
Grid search	2278	CR;32;3;1;2 CR;16;5;1;2 FS;32;0.4	96	218298
GA	1533	CT;32;5;2;3 FL;16;0.2	98	52218
GA + Predictor	1528	CRN;32;9;1;4 FL;64;0.4	98	595201

Table 6.5: CIFAR-10 summary.

Method	Evaluated solutions	Best solution		
		Model(s)	Acc (%)	Weights
Random search	4000	CT;16;5;2;4	50	20586
		CT;32;5;1;4	50	202442
		CT;16;7;1;4 FT;64;0.6	50	544778
Grid search	2754	CR;16;3;1;2 CR;32;5;1;2 FT;64;0.4	48	119364
GA	1733	CT;32;5;1;4 FT;64;0.6	53	1283146
GA + Predictor	975	CT;32;5;1;4 FS;64;0.4	53	1283146

training and test accuracy, when compared to the best model (LENET). These optimized solutions were obtained without human intervention, through the evolution of an initial population of randomized solutions. For CIFAR-10, the GANet model presented inferior performance, when compared to the benchmark results. The GA+PNet

Table 6.6: Optimized and reference models (MNIST).

Model	Descriptor	Acc (%)		Weights	$\Delta t(s)$
		Train	Test		
LENET	CL;20;5;1;2	100	98.8	8131080	373
	CL;50;5;1;2				
	FR;500;0.0				
C10	CR;32;5;1;3	99.1	97.6	488042	4121
	CR;32;5;1;3				
	CR;64;5;1;3				
	FL;64;0.0				
SPyr_Rev	CL;50;5;1;2	100	98.6	3271830	652
(LENET)	CL;20;5;1;2				
	FR;500;0.0				
SPyr_Rev	CR;64;5;1;3	99.4	98.3	284042	3136
(C10)	CR;32;5;1;3				
	CR;32;5;1;3				
	FL;64;0.0				
GANet	CT;32;5;2;3	99.6	98.3	52218	242
(MNIST)	FL;16;0.2				
GA+PNet	CRN;32;9;1;4	99.7	98.8	595201	429
(MNIST)	FL;64;0.4				

model showed slightly better results, with a decrease of 2-3% when compared to the benchmark average in the evaluated samples, despite requiring less training time than 2 of the benchmark models. However, as for the MNIST experiment, the proposed method's objective is to provide researches and practitioners with competitive starting points, without requiring prior expertise or intervention. These starting models can and should be subject to a fine tunning process, in order to improve their performance for the target problem. Also, the classification improvement observed in this last experiment, when compared to the optimization stage, indicate that more challenging datasets might require larger training sets or bigger populations of solutions during optimization, at the cost of higher usage of computational resources. Further work will be

Table 6.7: Optimized and reference models (CIFAR-10).

Model	Descriptor	Acc (%)		Weights	$\Delta t(s)$
		Train	Test		
LENET	CL;20;5;1;2	100	100	12132080	1514
	CL;50;5;1;2				
	FR;500;0.0				
C10	CR;32;5;1;3	97.8	98.6	882858	6953
	CR;32;5;1;3				
	CR;64;5;1;3				
	FL;64;0.0				
SPyr_Rev	CL;50;5;1;2	99.5	99.2	3271830	2005
(LENET)	CL;20;5;1;2				
	FR;500;0.0				
SPyr_Rev	CR;64;5;1;3	89.8	88.5	483850	13204
(C10)	CR;32;5;1;3				
	CR;32;5;1;3				
	FL;64;0.0				
GANet	CTN;32;5;1;4	75.6	75.1	1283146	3794
(Cifar10)	FT;64;0.6				
GA+PNet	CT;32;5;1;4	94.1	93.9	1283146	4870
(Cifar10)	FS;64;0.4				

conducted to investigate these hypotheses, as well as the improvement of the fitness predictor demonstrated in this work.

6.4 Evolutionary selection of deep visual trackers

In order to evaluate the performance of the evolutionary selection technique presented in the last section in deep visual tracking, the AR marker detection and localization datasets described in sections 6.2.1 and 6.2.2 were used. For marker detection, the same setup of the MNIST and CIFAR-10 experiments was used. Figure 6.13 presents the evolution of the best solution and population average for 20 generations, including

the initial population represented by 0-indexed generation.

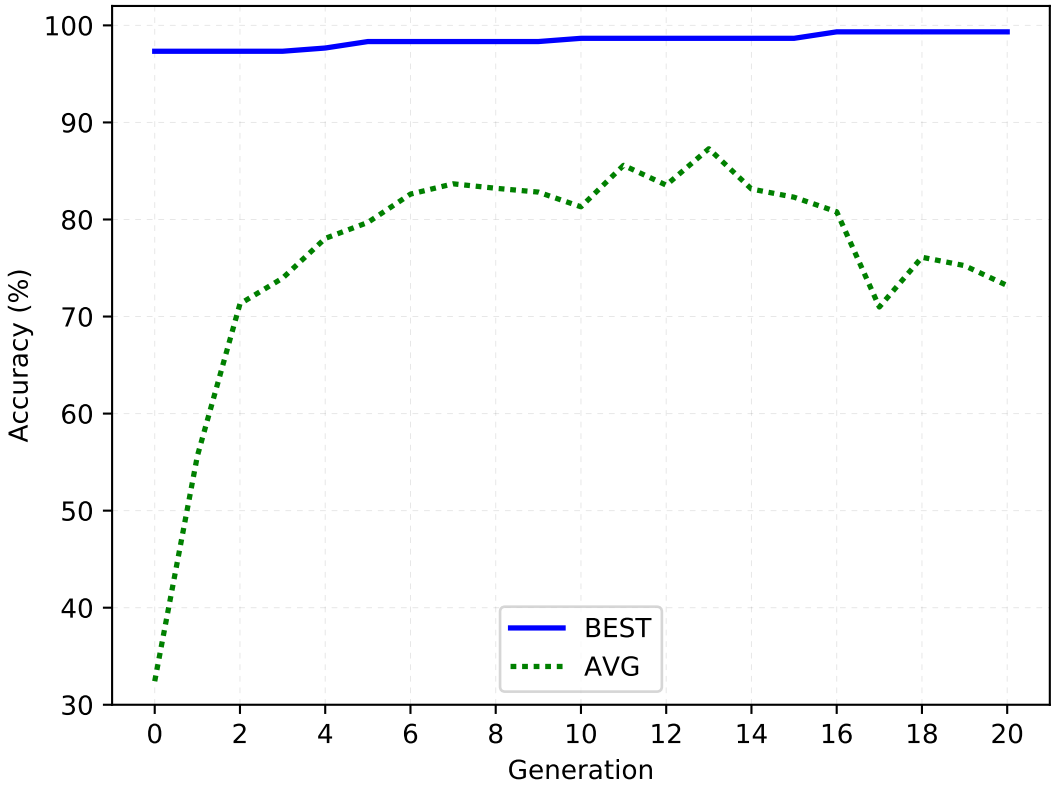


Figure 6.13: Population evolution for AR marker detection.

Although models with more than 90% of accuracy were found since the initial population, the best model was found by the GA at the 16th generation. Population average fitness increased until the 13th generation, after which a slight decrease was observed. This decrease is justified by the possible convergence of the population to a set of solutions whose combination results in poorly fitted individuals. Trough elitism, since the best individuals are kept for further generations, this behavior is observed until the 20th generation.

As observed in the experiments described in sections 6.2.1 and 6.2.2, the regression problem presents a slightly different error curve, when compared to image classification applications. Among others, this difference is justified by the distinct loss functions used in each domain (accuracy for classification and Euclidean distance for regression problems). When conducting a first evaluation of the DNN models applied to AR marker localization, a small but continuous improvement was observed until

the latest epochs for many models, increasing the need for computational resources since early sopping will rarely be reached. Therefore, the maximum number of training epochs was reduced by 10, with the best test loss used as the model fitness, as in the work of Assunção et al. (2018). An early stopping of 5 epochs, as well as the fitness prediction technique described in Section 5.5.4 were setup. Figure 6.14 presents the population evolution, considering the 10 training epochs for each model.

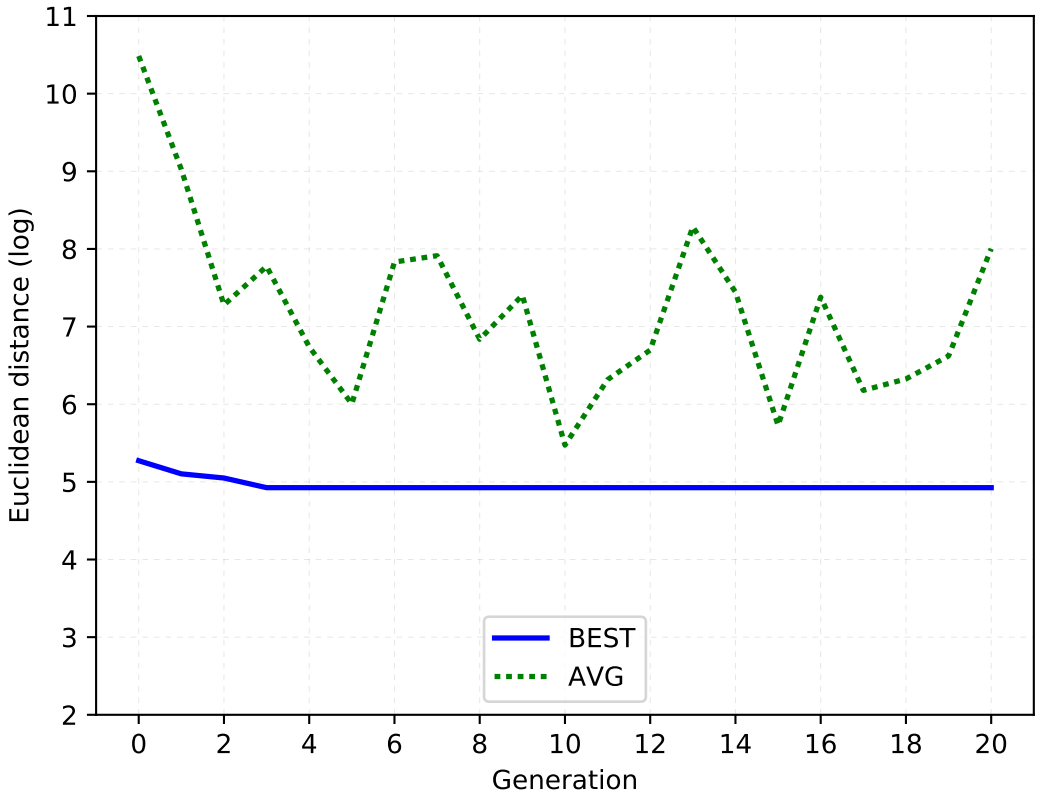


Figure 6.14: Population evolution: AR marker localization.

In this experiment, the best solution (found in the 3th generation) was not outperformed by any solution in the further generations. In addition, through the evolution of the average fitness, one can observe the lack of convergence of the evaluated solutions in this experiment. This can be explained by the lack of relevant information in the first epochs of training, since models are evaluated for only 10 epochs in this experiment. Therefore, the balance between available computational resources and expected performance should be considered for this class of application. Despite this behavior, the best models were subject to further analysis, in order to evaluate their

performance when more resources are available.

The 5 best solutions found by the GA in both detection and localization experiments were subject to further training, for 500 epochs, early stopping when no improvement was observed for 20 consecutive epochs. Tables 6.8 and 6.9 present the results of this evaluation, including model accuracy/loss for training and test samples, as well as the average frame processing delay (Δt_p). To provide fair comparisons, the models obtained with the grid search method were trained using the same parameters. These models presented slightly different values for the evaluated metrics, when compared to those described in Table 6.3, mainly because of the different number of epochs and early stopping conditions.

Table 6.8: Best models found in grid and GA search for marker detection. Models are trained for a maximum of 500 epochs, with an early stopping of 20 epochs.

Search method	**Descriptor**	**Acc (%)**		$\mathbf{\Delta t_p}$ **(ms)**
		Train	**Test**	
Grid search	FL;8;0-FR;4;0-FR;8;0	89.83	91.33	3.45
	CR;16;17;4;2-CL;4;11;4;0	92.67	94.67	24.14
	CRN;12;11;4;2-FL;8;0	97.58	94.33	17.88
GA + Predictor	CRN;4;11;4;2	99.25	96.00	9.26
	CR;4;11;4;2	99.42	98.67	6.83
	CRN;4;11;4;4	98.42	96.00	9.68
	CRN;4;11;4;3	99.50	94.67	8.77
	CR;4;11;4;3	99.75	97.00	6.62

DNN models optimized using GA search outperformed the models obtained through grid search, presenting equal or better accuracies for both training and test samples. In addition, processing delays make these models suitable for real time applications such as AR systems. The influence of evolutionary selection can be observed through the similarities of the resulting models, probably resulting from the application of genetic operators to fittest individuals, leading to population convergence around those individuals. In Table 6.9 the results for the localization dataset are summarized.

The GA + Predictor search technique was able to find competitive models, when compared to the grid-based approach described in Section 5.4. Some of the automatic

Table 6.9: Best models found in grid and GA search for marker localization. Models are trained for a maximum of 500 epochs, with an early stopping of 20 epochs.

Search method	**Descriptor**	**Train RMSE (px)**			**Test RMSE (px)**			Δt_p **(ms)**
		MIN	AVG (STD)	MAX	MIN	AVG (STD)	MAX	
Grid search	FL;8;0-FL;8;0	9.62	32.45 (12.89)	93.61	9.73	31.96 (13.76)	87.30	3.04
	CR;12;13;4;0-CL;8;11;4;0	1.35	5.97 (3.86)	26.05	1.96	6.20 (4.37)	26.11	12.71
	CR;20;9;4;0-FL;16;0	0.80	3.39 (1.96)	16.42	0.96	3.42 (2.08)	13.20	6.00
GA + Predictor	CTN;8;3;2;4	0.62	4.33 (2.52)	17.81	0.92	4.18 (2.29)	16.36	40.04
	CLN;16;3;3;2	0.63	4.34 (2.53)	17.83	0.93	4.18 (2.29)	16.36	40.01
	CT;4;7;3;3	0.85	4.36 (2.72)	20.09	1.00	4.58 (3.20)	21.58	7.05
	CL;4;5;2;2	16.15	55.14 (25.62)	124.86	17.49	59.20 (27.33)	120.79	6.01
	CL;8;5;2;2	33.82	127.51 (39.78)	266.30	40.14	125.82 (40.76)	261.75	8.93

generated models outperformed two of the grid-optimized models in almost all metrics, except for the processing time in some cases. As explained in the previous sections, it is important to notice, however, that these are starting models, obtained without human intervention and subject only to evolutionary optimization. Refinements on these models should be done to target specific application metrics, such as processing time or localization error reduction. In addition, a combination of both methods, in which automatically generated models are refined through grid search can lead to better results when compared to the use of a single procedure, while reducing the search space to a promising neighborhood.

6.5 Final considerations

This chapter presented the results of the experiments conducted in this work. The proposed architecture model were evaluated with 2 distinct proofs of concept (OpenCV/libVLC and WebRTC/JSARToolKit), demonstrating the integration of components from different vendors for AR distribution. Deep visual trackers were developed and compared with alternatives from the literature, revealing their potential application in AR maker detection and localization. Finally, an evolutionary method for deep neural network topology selection was proposed and evaluated, using two of the most widely known datasets for image processing applications, as well as the AR marker tracking dataset. In the following paragraphs, some limitations found in the described experiments will be discussed.

Although the designed deep visual trackers presented good accuracy on marker detection, their performance on marker localization should be further improved. The small dataset used (600 frames for training and 150 frames for testing) limited the number of training epochs, in order to prevent model overfitting. The use of larger datasets (or dataset augmentation techniques) can provide better results for marker tracking, by offering more heterogeneous training samples to the networks.

The computational cost associated with the GA selection technique made it prohibitive to exhaustively experiment different parameters for the GA itself. Given the vast search space of possible solutions, very different populations are generated if distinct parameters are used, making it difficult to reuse previously evaluated models. Although

fitness prediction significantly reduced the number of evaluated models, improvements should be investigated in order to reduce the computational cost associated with the evaluation of a single model from scratch. Performance prediction from early training steps, as well as weight sharing mechanisms are some possible contributions worth of investigation to overcome this limitation.

In the next chapter, final conclusions on the main contributions of this work, as well as future work suggestions will be presented.

Chapter 7

Conclusions and future work

In this work, an architecture model for distributed Augmented Reality was presented and evaluated using two distinct implementations: a desktop-based and a browser-based solution. The layered nature of both solutions favors modularization and ease of component replacement, which is important when experimenting similar implementations or adapting other technologies to an existing application. The designed architecture model made it possible to integrate software components from different providers in an AR distribution context, validating the first hypothesis investigated in this work. While the desktop solution presented better performance on resources usage, the browser-based solution favors portability, since it uses the browser as the targeted platform.

Implemented proofs of concept presented promising results on performance and image quality metrics. However, comparison of the obtained latency metrics with reference values from the literature show that improvements in this feature should be made, in order to match real-time requirements of common applications. Future investigation will be conducted to optimize the related components, toward the reduction of transmission delays.

The experiments designed to evaluate the use of deep learning techniques in AR marker tracking reveal their promising use, specially when applied to remotely captured videos subjected to network streaming. The developed trackers outperformed trackers found in the literature when applied to images deteriorated by network streaming, presenting a reduced loss of detection capability, and therefore validating the second hypothesis investigated in this thesis. Finally, processing delay (detection + localization)

of deep learning trackers makes them suitable for AR systems, enabling the desired rendering rate of 30 fps for real-time applications.

The evolutionary-based topology selection techniques presented in this work were able to achieve competitive results with minimal human intervention and without requiring previous domain knowledge, while using less computational resources when compared to random and grid search techniques. The optimized solutions were compared to state-of-the-art models when applied to the well know MNIST and CIFAR-10 datasets, offering promising starting points given the vast search space of deep neural network hyper-parameters. When applied to deep visual tracking, the models found with evolutionary search outperformed detectors optimized through grid search. For marker localization, competitive starting models were found during automatic search, requiring, however, a fine tunning stage in order to match the performance of trackers optimized with grid search. These results indicate the potential application of evolutionary computation in deep neural network topology selection, and confirm the third hypothesis addressed in this work.

The use of a fitness predictor in the model evaluation stage resulted in a significant reduction on the number of evaluated models and execution time for some of the explored datasets. This fact demonstrates the benefits of fitness prediction, but suggests that the use of other prediction methods, as well as the optimization of the predictor itself may lead to different results. Such methods should include accuracy prediction from early epochs, weight sharing and gradual increase of resources availability such as training samples and epochs.

For future work, the correlation between tracking performance and objective image quality metrics will be investigated. Assessment of the impact of dataset augmentation (for example, through the use of synthetic images) in deep visual trackers' performance will also be conducted. Finally, improvement on the evolutionary topology selection technique will be studied through the investigation of other alternatives to fitness prediction, and also the experimentation of different parameters for evolutionary pressure tunning.

Appendix A

Publications

The following list presents published work, accomplished during doctoral studies:

- CAETANO, D.; MATTIOLI, F.; LAMOUNIER E.; CARDOSO, A. On the use of augmented reality techniques in a telerehabilitation environment for wheelchair users' training. In: *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, 2014. https://doi.org/10.1109/ISMAR.2014.6948473.

- CAETANO, D.; MATTIOLI, F.; CARDOSO, A.; SOARES, A.; LAMOUNIER, E. O Uso de Realidade Aumentada em Treinamento de Cadeirantes por Telereabilitação (The Use of Augmented Reality and Telerehabilitation for Wheelchair Users Training). In: *Tendências e Técnicas em Realidade Virtual e Aumentada (Tendencies and Techniques in Virtual and Augmented Reality)*, 2015.

- MATTIOLI, F.; CAETANO, D.; CARDOSO, A.; LAMOUNIER, E. On the agile development of virtual reality systems. In: *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*, 2015.

- MATTIOLI, F; CAETANO, D; CARDOSO, A.; NAVES, E; LAMOUNIER, E. An Experiment on the Use of Genetic Algorithms for Topology Selection in Deep Learning. *Journal of Electrical and Computer Engineering*, Hindawi, v. 2019, 2019. https://doi.org/10.1155/2019/3217542.

References

[Akgul, Penekli and Genc 2016] AKGUL, O.; PENEKLI, H.; GENC, Y. Applying deep learning in augmented reality tracking. In: IEEE. *Signal-Image Technology & Internet-Based Systems (SITIS), 2016 12th Intl. Conference on*. 2016. p. 47–54. https://doi.org/10.1109/SITIS.2016.17.

[AlBasiouny, Sarhan and Medhat 2015] ALBASIOUNY, E. R.; SARHAN, A.; MEDHAT, T. Mean-shift-fast algorithm to handle motion-blur with tracking fiducial markers. In: IEEE. *Computer Engineering & Systems (ICCES), 2015 Tenth International Conference on*. 2015. p. 286–292. https://doi.org/10.1109/ICCES.2015.7393061.

[Albelwi and Mahmood 2017] ALBELWI, S.; MAHMOOD, A. A framework for designing the architectures of deep convolutional neural networks. *Entropy*, Multidisciplinary Digital Publishing Institute, v. 19, n. 6, 2017. https://doi.org/10.3390/e19060242.

[Ali and Simeone 2017] ALI, A.-S.; SIMEONE, O. Energy-efficient resource allocation for mobile edge computing-based augmented reality applications. *IEEE Wireless Communications Letters*, IEEE, 2017.

[Alvarado et al. 2018] ALVARADO, L. A. R. et al. Layered software architecture for the development of mobile learning objects with augmented reality. *IEEE Access*, IEEE, v. 6, p. 57897–57909, 2018. https://doi.org/10.1109/ACCESS.2018.2873976.

[Anand and Guha 1995] ANAND, S.; GUHA, S. Computer aided rehabilitation for the handicapped. In: *Engineering in Medicine and Biology Society, 1995 and 14th Conference of the Biomedical Engineering Society of India. An International Meeting, Proceedings of the First Regional Conference., IEEE*. [s.n.], 1995. p. 2/119–2/120. https://doi.org/10.1109/RCEMBS.1995.532187.

[Arcila et al. 2006] ARCILA, T. et al. Flowvr: A framework for distributed virtual reality applications. In: *Premières journées de l'Association Française de Réalité Virtuelle, Augmentée, Mixte et d'Interaction 3D*. Rocquencourt, France: [s.n.], 2006.

[ARToolKit 2019] ARToolKit. *ARToolKit.js*. 2019. Available from Internet: https://github.com/artoolkit/jsartoolkit5. Accessed: 2019-05-20.

[Assunção et al. 2018] ASSUNÇÃO, F. et al. Evolving the topology of large scale deep neural networks. In: SPRINGER. *European Conference on Genetic Programming*. 2018. p. 19–34. https://doi.org/10.1007/978-3-319-77553-1_2.

[Azuma 1997] AZUMA, R. T. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, MIT Press, v. 6, n. 4, p. 355–385, 1997. https://doi.org/10.1162/pres.1997.6.4.355.

[Bachmann et al. 2000] BACHMANN, F. et al. *Technical Concepts of Component-Based Software Engineering*. 2. ed. [S.l.], 2000. https://doi.org/10.21236/ADA379930.

[Baggio 2012] BAGGIO, D. L. *Mastering OpenCV with practical computer vision projects*. [S.l.]: Packt Publishing Ltd, 2012.

[Baker et al. 2016] BAKER, B. et al. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[Bauer et al. 2001] BAUER, M. et al. Design of a component-based augmented reality framework. In: IEEE. *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*. [S.l.], 2001. p. 45–54.

[Bauer et al. 2003] BAUER, M. et al. Integrating studierstube and dwarf. In: *Proceedings of International Workshop on Software Technology for Augmented Reality Systems (STARS 2003)*. [S.l.: s.n.], 2003. p. 1–5. TR-188-2-2003-21.

[Bender et al. 2018] BENDER, G. et al. Understanding and simplifying one-shot architecture search. In: *International Conference on Machine Learning*. [S.l.: s.n.], 2018. p. 549–558.

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

[Berretti et al. 2011] BERRETTI, S. et al. 3d facial expression recognition using sift descriptors of automatically detected keypoints. *The Visual Computer*, Springer, v. 27, n. 11, p. 1021–1036, 2011. https://doi.org/10.1007/s00371-011-0611-x.

[Billinghurst et al. 2015] BILLINGHURST, M. et al. A survey of augmented reality. *Foundations and Trends® Human–Computer Interaction*, Now Publishers, Inc., v. 8, n. 2-3, p. 73–272, 2015. https://doi.org/10.1561/1100000049.

[Billinghurst and Duenser 2012] BILLINGHURST, M.; DUENSER, A. Augmented reality in the classroom. *Computer*, IEEE, v. 45, n. 7, p. 56–63, 2012. https://doi.org/10.1109/MC.2012.111.

[Brennan et al. 2011] BRENNAN, D. M. et al. A telerehabilitation platform for home-based automated therapy of arm function. In: IEEE. *2011 Annual International Conference of Engineering in Medicine and Biology Society, EMBC*. 2011. p. 1819–1822. https://doi.org/10.1109/IEMBS.2011.6090518.

[Buduma and Locascio 2017] BUDUMA, N.; LOCASCIO, N. *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms*. [S.l.]: " O'Reilly Media, Inc.", 2017.

[Burdea 2004] BURDEA, G. C. Invited paper: Low-cost telerehabilitation. In: *Proceedings of Third International Workshop on Virtual Rehabilitation*. [S.l.: s.n.], 2004. p. 3297–3300.

[Buschmann, Henney and Schmidt 2007] BUSCHMANN, F.; HENNEY, K.; SCHMIDT, D. C. *Pattern-oriented software architecture: a pattern language for distributed computing*. [S.l.]: John wiley & sons, 2007.

[Caetano et al. 2015] CAETANO, D. et al. O uso de realidade aumentada em treinamento de cadeirantes por telereabilitação (the use of augmented reality and telerehabilitation for wheelchair users' training). In: *Tendências e Técnicas em Realidade*

Virtual e Aumentada (Tendencies and Techniques in Virtual and Augmented Reality). [S.l.: s.n.], 2015. p. 102–120.

[Caetano et al. 2014] CAETANO, D. et al. On the use of augmented reality techniques in a telerehabilitation environment for wheelchair users' training. In: *Mixed and Augmented Reality (ISMAR), 2014 IEEE International Symposium on*. [s.n.], 2014. p. 329–330. https://doi.org/10.1109/ISMAR.2014.6948473.

[Caetano et al. 2018] CAETANO, D. et al. Adaptação de uma interface usb para joystick vr2 aplicada ao treinamento de usuários de cadeira de rodas (adaptation of an usb interface for vr2 joystick applied to wheelchair users' training). In: *Tecnologia Assistiva: Desenvolvimento e Aplicação (Assistive Technology: Development and Application)*. [S.l.: s.n.], 2018. p. 261–268.

[Cai, Zhu and Han 2018] CAI, H.; ZHU, L.; HAN, S. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

[Cardoso and Lamounier 2006] CARDOSO, A.; LAMOUNIER, E. *Fundamentos e tecnologia de Realidade Virtual e Aumentada*. Porto Alegre: Editora SBC, 2006. A Realidade Virtual na Educação e Treinamento.

[Chikkerur et al. 2011] CHIKKERUR, S. et al. Objective video quality assessment methods: A classification, review, and performance comparison. *Broadcasting, IEEE Transactions on*, v. 57, n. 2, p. 165–182, June 2011. ISSN 0018-9316. https://doi.org/10.1109/TBC.2011.2104671.

[Chouiten, Didier and Mallem 2011] CHOUITEN, M.; DIDIER, J.-Y.; MALLEM, M. Component-based middleware for distributed augmented reality applications. In: ACM. *Proceedings of the 5th International Conference on Communication System Software and Middleware*. 2011. p. 3. https://doi.org/10.1145/2016551.2016554.

[Chouiten, Didier and Mallem 2012] CHOUITEN, M.; DIDIER, J.-Y.; MALLEM, M. Distributed augmented reality systems: How much performance is enough? In: *Multimedia and Expo Workshops (ICMEW), 2012 IEEE International Conference on*. [s.n.], 2012. p. 337–342. https://doi.org/10.1109/ICMEW.2012.64.

[Crivellaro et al. 2015] CRIVELLARO, A. et al. A novel representation of parts for accurate 3d object detection and tracking in monocular images. In: *Proceedings of the IEEE International Conference on Computer Vision*. [s.n.], 2015. p. 4391–4399. https://doi.org/10.1109/ICCV.2015.499.

[Cuccu 2018] CUCCU, G. *Extending the Applicability of Neuroevolution*. PhD thesis — University of Fribourg (Switzerland), 2018.

[Deutsch, Lewis and Burdea 2007] DEUTSCH, J. E.; LEWIS, J. A.; BURDEA, G. Technical and patient performance using a virtual reality-integrated telerehabilitation system: preliminary finding. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, IEEE, v. 15, n. 1, p. 30–35, 2007. https://doi.org/10.1109/TNSRE.2007.891384.

[Dhurjaty 2004] DHURJATY, S. The economics of telerehabilitation. *Telemedicine Journal & E-Health*, Mary Ann Liebert, Inc., v. 10, n. 2, p. 196–199, 2004. https://doi.org/10.1089/tmj.2004.10.196.

[Didier et al. 2012] DIDIER, J. Y. et al. Arcs: A framework with extended software integration capabilities to build augmented reality applications. In: *2012 5th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. [s.n.], 2012. p. 60–67. ISSN 2328-7772. https://doi.org/10.1109/SEARIS.2012.6231170.

[Dobrica and Niemela 2002] DOBRICA, L.; NIEMELA, E. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering*, v. 28, n. 7, p. 638–653, Jul 2002. ISSN 0098-5589. https://doi.org/10.1109/TSE.2002.1019479.

[Duchi, Hazan and Singer 2011] DUCHI, J.; HAZAN, E.; SINGER, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, v. 12, n. Jul, p. 2121–2159, 2011.

[Dutta and Gros 2018] DUTTA, S.; GROS, E. Evaluation of the impact of deep learning architectural components selection and dataset size on a medical imaging task. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Medical Imaging 2018: Imaging Informatics for Healthcare, Research, and Applications*. 2018. v. 10579, p. 1057911. https://doi.org/10.1117/12.2293395.

[Engel, Schöps and Cremers 2014] ENGEL, J.; SCHÖPS, T.; CREMERS, D. Lsd-slam: Large-scale direct monocular slam. In: SPRINGER. *European Conference on Computer Vision*. 2014. p. 834–849. https://doi.org/10.1007/978-3-319-10605-2_54.

[Engelke et al. 2013] ENGELKE, T. et al. Content first - a concept for industrial augmented reality maintenance applications using mobile devices. In: *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. [s.n.], 2013. p. 251–252. https://doi.org/10.1109/ISMAR.2013.6671790.

[Fausett et al. 1994] FAUSETT, L. V. et al. *Fundamentals of neural networks: architectures, algorithms, and applications*. 3. ed. [S.l.]: Prentice-Hall Englewood Cliffs, 1994.

[Forsman, Arvo and Lehtonen 2016] FORSMAN, M.; ARVO, J.; LEHTONEN, T. Extended panorama tracking algorithm for augmenting virtual 3d objects in outdoor environments. In: IEEE. *Virtual System & Multimedia (VSMM), 2016 22nd International Conference on*. 2016. p. 1–8. https://doi.org/10.1109/VSMM.2016.7863190.

[Gamma et al. 1994] GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Pearson Education, 1994.

[Garon and Lalonde 2017] GARON, M.; LALONDE, J.-F. Deep 6-dof tracking. *IEEE transactions on visualization and computer graphics*, IEEE, v. 23, n. 11, p. 2410–2418, 2017. https://doi.org/10.1109/TVCG.2017.2734599.

[Gonnet et al. 2001] GONNET, S. et al. An environment for modeling and managing the process design process. *Latin American Applied Research*, v. 31, p. 419–425, 2001.

[Goodfellow, Bengio and Courville 2016] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Available from Internet: http://www.deeplearningbook.org. Accessed: 2019-05-20.

[Haykin 2009] HAYKIN, S. *Neural networks and learning machines*. 3. ed. [S.l.]: Upper Saddle River: Pearson Education, 2009.

[Hinterstoisser et al. 2012] HINTERSTOISSER, S. et al. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern*

Analysis and Machine Intelligence, IEEE, v. 34, n. 5, p. 876–888, 2012. https://doi.org/10.1109/TPAMI.2011.206.

[Hinton, Osindero and Teh 2006] HINTON, G. E.; OSINDERO, S.; TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation*, MIT Press, v. 18, n. 7, p. 1527–1554, 2006. https://doi.org/10.1162/neco.2006.18.7.1527.

[Hinton et al. 2012] HINTON, G. E. et al. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[Holland 1992] HOLLAND, J. H. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992. https://doi.org/10.7551/mitpress/1090.001.0001.

[Hondori et al. 2013] HONDORI, H. M. et al. A spatial augmented reality rehab system for post-stroke hand rehabilitation. *Studies in Health Technology and Informatics*, IOS Press, v. 184, n. 20, p. 279–285, 2013.

[Hore and Ziou 2013] HORE, A.; ZIOU, D. Is there a relationship between peak-signal-to-noise ratio and structural similarity index measure? *Image Processing, IET*, v. 7, n. 1, p. 12–24, February 2013. ISSN 1751-9659. https://doi.org/10.1049/iet-ipr.2012.0489.

[Huang et al. 2018] HUANG, S. et al. Gnas: A greedy neural architecture search method for multi-attribute learning. *arXiv preprint arXiv:1804.06964*, 2018. https://doi.org/10.1145/3240508.3240588.

[Huang et al. 2013] HUANG, Z. et al. Mobile augmented reality survey: a bottom-up approach. *arXiv preprint 1309.4413*, 2013.

[Huang et al. 2014] HUANG, Z. et al. Cloudridar: A cloud-based architecture for mobile augmented reality. In: ACM. *Proceedings of the 2014 workshop on Mobile augmented reality and robotic technology-based systems*. 2014. p. 29–34. https://doi.org/10.1145/2609829.2609832.

[Hughes et al. 2005] HUGHES, C. E. et al. Mixed reality in education, entertainment, and training. *IEEE computer graphics and applications*, IEEE, v. 25, n. 6, p. 24–30, 2005. https://doi.org/10.1109/MCG.2005.139.

[Ibanez et al. 2016] IBANEZ, M.-B. et al. Support for augmented reality simulation systems: the effects of scaffolding on learning outcomes and behavior patterns. *IEEE Transactions on Learning Technologies*, IEEE, v. 9, n. 1, p. 46–56, 2016. https://doi.org/10.1109/TLT.2015.2445761.

[Ines and Abdelkader 2011] INES, D. L.; ABDELKADER, G. Mixed reality serious games: The therapist perspective. In: *Proceedings of the 2011 IEEE 1st International Conference on Serious Games and Applications for Health*. Washington, DC, USA: IEEE Computer Society, 2011. (SEGAH '11), p. 1–10. ISBN 978-1-4673-0433-7. http://dx.doi.org/10.1109/SeGAH.2011.6165462.

[Irawati et al. 2008] IRAWATI, S. et al. Varu framework: Enabling rapid prototyping of vr, ar and ubiquitous applications. In: IEEE. *2008 IEEE Virtual Reality Conference*. 2008. p. 201–208. https://doi.org/10.1109/VR.2008.4480774.

[Jarrett et al. 2009] JARRETT, K. et al. What is the best multi-stage architecture for object recognition? In: IEEE. *Computer Vision, 2009 IEEE 12th International Conference on*. 2009. p. 2146–2153. https://doi.org/10.1109/ICCV.2009.5459469.

[Jia et al. 2014] JIA, Y. et al. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. https://doi.org/10.1145/2647868.2654889.

[Jong 2006] JONG, K. A. D. *Evolutionary computation: a unified approach*. [S.l.]: MIT press, 2006.

[Kerrisk 2019] KERRISK, M. *NetEm - Network Emulator*. 2019. Available from Internet: http://man7.org/linux/man-pages/man8/tc-netem.8.html. Accessed: 2019-05-20.

[Kim et al. 2018] KIM, K. et al. Revisiting trends in augmented reality research: A review of the 2nd decade of ismar (2008–2017). *IEEE transactions on visualization and computer graphics*, IEEE, v. 24, n. 11, p. 2947–2962, 2018. https://doi.org/10.1109/TVCG.2018.2868591.

[Kingma and Ba 2014] KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[Kinsella 1999] KINSELLA, A. Current issues in design of home telecare technologies. In: *Proceedings of the 21st Annual Conference and the 1999 Annual Fall Meetring of the Biomedical Engineering Society*. [S.l.]: IEEE, 1999.

[Klaue, Rathke and Wolisz 2003] KLAUE, J.; RATHKE, B.; WOLISZ, A. Evalvid–a framework for video transmission and quality evaluation. In: SPRINGER. *International conference on modelling techniques and tools for computer performance evaluation*. 2003. p. 255–272. https://doi.org/10.1007/978-3-540-45232-4_16.

[Krizhevsky and Hinton 2009] KRIZHEVSKY, A.; HINTON, G. *Learning multiple layers of features from tiny images*. [S.l.], 2009.

[Krizhevsky, Sutskever and Hinton 2012] KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2012. p. 1097–1105.

[Lathan 2000] LATHAN, G. Dimensions of diversity in design of telerehabilitation systems for universal usability. In: *Proceedings on the 2000 Conference on Universal Usability*. New York, NY, USA: ACM, 2000. (CUU '00), p. 61–62. ISBN 1-58113-314-6. Chairman-Rosen, M. and Chairman-Brennan, D. and Chairman-Trepagnier, C. and Chairman-Tran, B. and Chairman-Lauderdale, D. http://doi.acm.org/10.1145/355460.355473.

[Layaida and Hagimont 2005] LAYAIDA, O.; HAGIMONT, D. Plasma: a component-based framework for building self-adaptive multimedia applications. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Electronic Imaging 2005*. 2005. p. 185–196. https://doi.org/10.1117/12.592148.

[LeCun, Bengio and Hinton 2015] LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. *Nature*, Nature Research, v. 521, n. 7553, p. 436–444, 2015. hrefhttps://doi.org/10.1038/nature14539https://doi.org/10.1038/nature14539.

[LeCun et al. 1998] LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, IEEE, v. 86, n. 11, p. 2278–2324, 1998. hrefhttps://doi.org/10.1109/5.726791https://doi.org/10.1109/5.726791.

[Leung et al. 2003] LEUNG, F. H.-F. et al. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural networks*, IEEE, v. 14, n. 1, p. 79–88, 2003. https://doi.org/10.1109/TNN.2002.804317.

[Li et al. 2018] LI, P. et al. Deep visual tracking: Review and experimental comparison. *Pattern Recognition*, Elsevier, v. 76, p. 323–338, 2018. https://doi.org/10.1016/j.patcog.2017.11.007.

[Lima et al. 2017] LIMA, J. et al. Markerless tracking system for augmented reality in the automotive industry. *Expert Systems with Applications*, Elsevier, 2017.

[Lin et al. 2017] LIN, S. et al. Ubii: Physical world interaction through augmented reality. *IEEE Transactions on Mobile Computing*, IEEE, v. 16, n. 3, p. 872–885, 2017. https://doi.org/10.1109/TMC.2016.2567378.

[Lopes et al. 2014] LOPES, R. de A. et al. A proposal for stress management using serious games associated to virtual and augmented reality. *Journal of Systemics*, Directory of Open Access Journals, v. 12, n. 3, p. 1–7, 2014.

[MacWilliams et al. 2003] MacWilliams, A. et al. Herding sheep: live system for distributed augmented reality. In: *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.* [s.n.], 2003. p. 123–132. https://doi.org/10.1109/ISMAR.2003.1240695.

[Mattioli et al. 2015] MATTIOLI, F. et al. On the agile development of virtual reality systems. In: *Proceedings of the International Conference on Software Engineering Research and Practice (SERP)*. [S.l.: s.n.], 2015. p. 10.

[Mattioli et al. 2019] MATTIOLI, F. et al. An experiment on the use of genetic algorithms for topology selection in deep learning. *Journal of Electrical and Computer Engineering*, Hindawi, v. 2019, 2019.

[Meer 2014] MEER, R. van der. Recent developments in computer assisted rehabilitation environments. *Military Medical Research*, BioMed Central, v. 1, n. 1, 2014. https://doi.org/10.1186/2054-9369-1-22.

[Migliorini, Mingozzi and Vallati 2011] MIGLIORINI, D.; MINGOZZI, E.; VALLATI, C. Performance evaluation of h.264/svc video streaming over mobile wimax. *Comput. Netw.*, Elsevier North-Holland, Inc., New York, NY, USA, v. 55, n. 15, p. 3578–3591, out. 2011. ISSN 1389-1286. http://dx.doi.org/10.1016/j.comnet.2011.07.012.

[Mitchell 1998] MITCHELL, M. *An introduction to genetic algorithms*. [S.l.]: MIT press, 1998.

[Moorthy and Bovik 2011] MOORTHY, A.; BOVIK, A. Blind image quality assessment: From natural scene statistics to perceptual quality. *Image Processing, IEEE Transactions on*, v. 20, n. 12, p. 3350–3364, Dec 2011. ISSN 1057-7149. https://doi.org/10.1109/TIP.2011.2147325.

[Mossel et al. 2012] MOSSEL, A. et al. Artifice-augmented reality framework for distributed collaboration. *International Journal of Virtual Reality*, v. 11, n. 3, p. 1–7, 2012.

[Murphy-Chutorian and Trivedi 2010] MURPHY-CHUTORIAN, E.; TRIVEDI, M. M. Head pose estimation and augmented reality tracking: An integrated system and evaluation for monitoring driver awareness. *IEEE Transactions on intelligent transportation systems*, IEEE, v. 11, n. 2, p. 300–311, 2010. https://doi.org/10.1109/TITS.2010.2044241.

[Naqvi et al. 2015] NAQVI, N. et al. To cloud or not to cloud: a context-aware deployment perspective of augmented reality mobile applications. In: ACM. *Proc. of the 30th Annual ACM Symposium on Applied Computing*. 2015. https://doi.org/10.1145/2695664.2695880.

[Newman et al. 2004] NEWMAN, J. et al. Ubiquitous tracking for augmented reality. In: IEEE. *Third IEEE and ACM International Symposium on Mixed and Augmented Reality*. [S.l.], 2004. p. 192–201.

[Object Management Group® 2019] Object Management Group®. *Common Object Request Broker Architecture™*. 2019. Available from Internet: https://www.corba.org. Accessed: 2019-05-20.

[Ohlenburg et al. 2004] OHLENBURG, J. et al. The morgan framework: enabling dynamic multi-user ar and vr projects. In: ACM. *Proceedings of the ACM*

symposium on Virtual reality software and technology. 2004. p. 166–169. https://doi.org/10.1145/1077534.1077568.

[Oliveira et al. 2016] OLIVEIRA, L. C. d. et al. Mobile augmented reality enhances indoor navigation for wheelchair users. *Research on Biomedical Engineering*, SciELO Brasil, v. 32, n. 2, p. 111–122, 2016. https://doi.org/10.1590/2446-4740.01515.

[Olson 2011] OLSON, E. AprilTag: A robust and flexible visual fiducial system. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2011. p. 3400–3407. https://doi.org/10.1109/ICRA.2011.5979561.

[OpenCV team 2019] OpenCV team. *OpenCV*. 2019. Available from Internet: https://opencv.org. Accessed: 2019-05-20.

[Ortiz-Catalan et al. 2014] ORTIZ-CATALAN, M. et al. Virtual reality. In: *Emerging Therapies in Neurorehabilitation*. Springer, 2014. p. 249–265. https://doi.org/10.1007/978-3-642-38556-8_13.

[Pani et al. 2014] PANI, D. et al. A device for local or remote monitoring of hand rehabilitation sessions for rheumatic patients. *Translational Engineering in Health and Medicine, IEEE Journal of*, v. 2, p. 1–11, 2014. ISSN 2168-2372. https://doi.org/10.1109/JTEHM.2014.2299274.

[Park and Park 2010] PARK, H.; PARK, J. Invisible marker–based augmented reality. *Intl. Journal of Human–Computer Interaction*, Taylor & Francis, v. 26, n. 9, 2010. https://doi.org/10.1080/10447318.2010.496335.

[Patterson and Gibson 2017] PATTERSON, J.; GIBSON, A. *Deep Learning: A practitioner's approach*. 1. ed. Sebastopol, CA, USA: O'Reilly Media, 2017. ISBN 9781491914250.

[Piekarski and Thomas 2003] PIEKARSKI, W.; THOMAS, B. H. An object-oriented software architecture for 3d mixed reality applications. In: IEEE. *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.* [S.l.], 2003. p. 247–256.

[Popescu et al. 2000] POPESCU, V. G. et al. A virtual-reality-based telerehabilitation system with force feedback. *IEEE Transactions on Information Technology in Biomedicine*, v. 4, n. 1, p. 45–51, 2000. https://doi.org/10.1109/4233.826858.

[Postolache et al. 2011] POSTOLACHE, G. et al. Rehabilitative telehealthcare for post-stroke outcome assessment. In: IEEE. *Pervasive Computing Technologies for Healthcare (PervasiveHealth), 2011 5th International Conference on*. 2011. p. 408–413. https://doi.org/10.4108/icst.pervasivehealth.2011.246141.

[Powell, Powell and Simmonds 2014] POWELL, W.; POWELL, V.; SIMMONDS, M. Virtual reality for gait rehabilitation - promises, proofs and preferences. In: *Proceedings of the 7th International Conference on PErvasive Technologies Related to Assistive Environments*. New York, NY, USA: ACM, 2014. (PETRA '14), p. 16:1–16:7. ISBN 978-1-4503-2746-6. http://doi.acm.org/10.1145/2674396.2674450.

[Prechelt 1998] PRECHELT, L. Early stopping-but when? In: *Neural Networks: Tricks of the trade*. Springer, 1998. p. 55–69. https://doi.org/10.1007/3-540-49430-8_3.

[Prisacariu and Reid 2012] PRISACARIU, V. A.; REID, I. D. Pwp3d: Real-time segmentation and tracking of 3d objects. *International journal of computer vision*, Springer, v. 98, n. 3, p. 335–354, 2012. https://doi.org/10.1007/s11263-011-0514-3.

[Real et al. 2017] REAL, E. et al. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[Reicher 2004] REICHER, T. *A framework for dynamically adaptable augmented reality systems*. PhD thesis — Technische Universität München, 2004.

[Reni et al. 1991] RENI, G. et al. A computer assisted rehabilitation experience in neuropsychologically disabled children. In: *Engineering in Medicine and Biology Society, 1991. Vol.13: 1991., Proceedings of the Annual International Conference of the IEEE*. [s.n.], 1991. p. 1155–1156. https://doi.org/10.1109/IEMBS.1991.684389.

[Richards 2015] RICHARDS, M. *Software architecture patterns*. [S.l.]: O'Reilly Media, Incorporated, 2015.

[Rincon et al. 2018] RINCON, A. L. et al. Evolutionary optimization of convolutional neural networks for cancer mirna biomarkers classification. *Applied Soft Computing*, Elsevier, 2018.

[Rublee et al. 2011] RUBLEE, E. et al. Orb: An efficient alternative to sift or surf. In: IEEE. *Computer Vision (ICCV), 2011 IEEE international conference on*. 2011. p. 2564–2571. https://doi.org/10.1109/ICCV.2011.6126544.

[Russel and Norvig 2009] RUSSEL, S.; NORVIG, P. *Artificial intelligence: A modern approach*. 3. ed. [S.l.]: Pearson, 2009.

[Safarik et al. 2018] SAFARIK, J. et al. Genetic algorithm for automatic tuning of neural network hyperparameters. In: INTERNATIONAL SOCIETY FOR OPTICS AND PHOTONICS. *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*. 2018. v. 10643, p. 106430Q. https://doi.org/10.1117/12.2304955.

[Schmalstieg et al. 2002] SCHMALSTIEG, D. et al. The studierstube augmented reality project. *Presence: Teleoperators & Virtual Environments*, MIT Press, v. 11, n. 1, p. 33–54, 2002. https://doi.org/10.1162/105474602317343640.

[Schulzrinne, Rao and Lanphier 1998] SCHULZRINNE, H.; RAO, A.; LANPHIER, R. *Real Time Streaming Protocol*. 1998. Available from Internet: http://www.ietf.org/rfc/rfc2326.txt. Accessed: 2019-05-20. https://doi.org/10.17487/rfc2326.

[Silva 2011] SILVA, M. D. *Realidade aumentada distribuída na Web com independência de servidor para manipulação de objetos virtuais na camada cliente*. Master's thesis — Universidade Federal de Uberlândia, Uberlândia, April 2011.

[Smithson et al. 2016] SMITHSON, S. C. et al. Neural networks designing neural networks: multi-objective hyper-parameter optimization. In: IEEE. *Computer-Aided Design (ICCAD), 2016 IEEE/ACM International Conference on*. 2016. p. 1–8. https://doi.org/10.1145/2966986.2967058.

[Sood et al. 2019] SOOD, A. et al. Neunets: An automated synthesis engine for neural network design. *arXiv preprint arXiv:1901.06261*, 2019.

[Srivastava et al. 2014] SRIVASTAVA, N. et al. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, JMLR. org, v. 15, n. 1, p. 1929–1958, 2014.

[Stanley et al. 2019] STANLEY, K. O. et al. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, Nature Publishing Group, v. 1, n. 1, p. 24–35, 2019. https://doi.org/10.1038/s42256-018-0006-z.

[Stanley and Miikkulainen 2002] STANLEY, K. O.; MIIKKULAINEN, R. Evolving neural networks through augmenting topologies. *Evolutionary computation*, MIT Press, v. 10, n. 2, p. 99–127, 2002. https://doi.org/10.1162/106365602320169811.

[Suganuma, Shirakawa and Nagao 2017] SUGANUMA, M.; SHIRAKAWA, S.; NAGAO, T. A genetic programming approach to designing convolutional neural network architectures. In: ACM. *Proceedings of the Genetic and Evolutionary Computation Conference*. 2017. p. 497–504. https://doi.org/10.1145/3071178.3071229.

[Sugomori et al. 2017] SUGOMORI, Y. et al. *Deep Learning: Practical Neural Networks with Java*. [S.l.]: Packt Publishing Ltd, 2017.

[Sun, Wang and Tang 2013] SUN, Y.; WANG, X.; TANG, X. Deep convolutional network cascade for facial point detection. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. [s.n.], 2013. p. 3476–3483. https://doi.org/10.1109/CVPR.2013.446.

[Szyperski 2002] SZYPERSKI, C. *Component Software: Beyond Object-Oriented Programming.* [S.l.: s.n.], 2002.

[Tayama, Hagino and Okada 2014] TAYAMA, Y.; HAGINO, M.; OKADA, K. I. Using a human-shaped input device for telerehabilitation. In: *Healthcare Informatics (ICHI), 2014 IEEE International Conference on*. [s.n.], 2014. p. 135–143. https://doi.org/10.1109/ICHI.2014.25.

[Tetteroo et al. 2015] TETTEROO, D. et al. Lessons learnt from deploying an end-user development platform for physical rehabilitation. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. New

York, NY, USA: ACM, 2015. (CHI '15), p. 4133–4142. ISBN 978-1-4503-3145-6. http://doi.acm.org/10.1145/2702123.2702504.

[The Qt Company 2019] The Qt Company. *Qt*. 2019. Available from Internet: https://www.qt.io. Accessed: 2019-05-20.

[The UNIX and Linux Forums 2019] The UNIX and Linux Forums. *Man page for ps*. 2019. Available from Internet: https://www.unix.com/man-page/All/1b/ps. Accessed: 2019-05-20.

[three.js 2019] three.js. *three.js*. 2019. Available from Internet: https://threejs.org. Accessed: 2019-05-20.

[Tokunaga et al. 2003] TOKUNAGA, E. et al. Object-oriented middleware infrastructure for distributed augmented reality. In: IEEE. *Object-Oriented Real-Time Distributed Computing, 2003. Sixth IEEE International Symposium on*. [S.l.], 2003. p. 156–163.

[Tokunaga et al. 2004] TOKUNAGA, E. et al. A middleware infrastructure for building mixed reality applications in ubiquitous computing environments. In: IEEE. *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services, 2004. MOBIQUITOUS 2004.* [S.l.], 2004. p. 382–391.

[Ullah and Petrosino 2016] ULLAH, I.; PETROSINO, A. About pyramid structure in convolutional neural networks. In: IEEE. *Neural Networks (IJCNN), 2016 International Joint Conference on*. 2016. p. 1318–1324. https://doi.org/10.1109/IJCNN.2016.7727350.

[VideoLAN organization 2019] VideoLAN organization. *libVLC*. 2019. Available from Internet: https://www.videolan.org/vlc/libvlc.html. Accessed: 2019-05-20.

[Walton et al. 2004] WALTON, J. et al. *NAG's IRIS explorer*. Citeseer, 2004. 633–654 p. https://doi.org/10.1016/B978-012387582-2/50034-4.

[Wang and Qian 2005] WANG, A. J. A.; QIAN, K. *Component-oriented programming*. John Wiley & Sons, 2005. https://doi.org/10.1002/0471713708.

[Wang et al. 2004] WANG, Z. et al. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, v. 13, n. 4, p. 600–612, April 2004. ISSN 1057-7149. https://doi.org/10.1109/TIP.2003.819861.

[Wang and Li 2011] WANG, Z.; LI, Q. Information content weighting for perceptual image quality assessment. *Image Processing, IEEE Transactions on*, v. 20, n. 5, p. 1185–1198, May 2011. ISSN 1057-7149. https://doi.org/10.1109/TIP.2010.2092435.

[WebRTC 2019] WebRTC. *WebRTC*. 2019. Available from Internet: https://webrtc.org. Accessed: 2019-05-20.

[Welchowski and Schmid 2016] WELCHOWSKI, T.; SCHMID, M. A framework for parameter estimation and model selection in kernel deep stacking networks. *Artificial intelligence in medicine*, Elsevier, v. 70, p. 31–40, 2016. https://doi.org/10.1016/j.artmed.2016.04.002.

[Wiederhold and Wiederhold 2014] WIEDERHOLD, M. D.; WIEDERHOLD, B. K. *Augmented Reality: What is it and How is it Enhancing Healthcare Today?* 2014. Available from Internet: http://www.cybertherapyandrehabilitation.com. Accessed: 2019-05-20.

[Xin et al. 2011] XIN, G. et al. 3d augmented reality teleoperated robot system based on dual vision. *The Journal of China Universities of Posts and Telecommunications*, Elsevier, v. 18, n. 1, p. 105–112, 2011. https://doi.org/10.1016/S1005-8885(10)60035-0.

[Yang, Zhao and Chan 2017] YANG, J.; ZHAO, Y.-Q.; CHAN, J. C.-W. Learning and transferring deep joint spectral–spatial features for hyperspectral classification. *IEEE Transactions on Geoscience and Remote Sensing*, IEEE, v. 55, n. 8, p. 4729–4742, 2017. https://doi.org/10.1109/TGRS.2017.2698503.

[Zhang et al. 2014] ZHANG, M. et al. A failure-aware explicit shape regression model for facial landmark detection in video. *IEEE Signal Processing Letters*, IEEE, v. 21, n. 2, p. 244–248, 2014. https://doi.org/10.1109/LSP.2013.2295231.

[Zhou, Duh and Billinghurst 2008] ZHOU, F.; DUH, H. B.-L.; BILLINGHURST, M. Trends in augmented reality tracking, interaction and display: A review of ten years of ismar. In: IEEE COMPUTER SOCIETY. *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*. [S.l.], 2008. p. 193–202.