technische universität
dortmund

Deep Face Recognition

Master Thesis (Revised)

Wilmar Fernando Moya Rueda
May 2, 2017

Supervisors:
Prof. Dr.-Ing. Gernot A. Fink
René Grzeszick, M.Sc.

Fakultät für Informatik
Technische Universität Dortmund
http://www.cs.uni-dortmund.de

CONTENTS

1 INTRODUCTION

Face recognition is a specific case of object recognition. It has received special attention in the recent years due to a great variety of applications such as robot-human interaction, control by gesture, surveillance, security, and people tracking. The idea of face recognition is to give a computer system the ability of finding and recognizing human faces fast and precisely in images or videos. Face recognition aspires to work similar to human perception. Humans identify a big number of known faces, even after years of separation, or under extreme occlusion conditions, e.g. just by looking at a small part of a face [TP91]. It is a complex task since faces can have different colors, poses, expressions, and sizes or they can be affected by illumination variations or occlusion conditions [AR15]. Today, there are different methods of face recognition: feature-based approaches (low-level analysis, feature analysis, active shape models) that make explicit use of facial-features such as skin color, facial landmarks or face geometry [HL01], and image-based approaches (linear subspace models, neural networks, statistical approaches) [YKA02].

The recent increase in the volume of data and computational resources has led to the need for fast and scalable recognition techniques. These techniques should be robust to non-rigid deformations, clutter, occlusion and illumination variations, but, at the same time, they must be sensitive to variations among faces from different persons. For these reasons, neural networks have become a surge of interest [TYRW14]. Initially, shallow regular neural networks could be used for small image sizes, but they would not scale to deeper networks since a huge amount of parameters would be necessary to be learnt, which can easily cause overfitting [FFL16]. As a result, Convolutional neural networks (CNNs) were proposed. In a CNN, neurons in a layer are connected to small regions of previous layers, which is different to regular neural networks, where all neurons are connected in a full manner. Besides, CNNs make correct assumptions about the nature of the images, for example locality of pixel dependencies [AK12]. In general, CNNs have fewer connections and parameters, they are easier to train, and they present similar or better performance than the regular neural networks [AK12]. As an extension of the CNNs, deep CNNs were introduced by [AK12] because a big number of high-resolution images and powerful GPUs are now available. The authors increased the number of convolutional layers, and they use large receptive fields in the first convolutional layer. As a result, overfitting, which is inherent to the large size

of the model, is avoided and effective results are achieved. In addition, the authors in [SZ15] continued adding convolutional layers, but they keep the size of the receptive fields very small (3x3 convolutions) with a stride of 1, throughout the whole network, which decreases the amount of parameters.

CNNs classify their input into classes; concretely for this thesis, they classify a set face-images into their identities, or map their input into a compact representation for verification purposes, where two face-images are compared for answering whether they show the same identity. Deep CNN architectures are used for solving the aforementioned tasks, following certain guidelines for training and testing, using techniques for avoiding overfitting, that were designed for general object recognition. However, some of the guidelines are not suitable for, specifically, face recognition and verification purposes. Besides, extreme-occlusion conditions in previous works have not been completely studied. Benchmark datasets for face recognition and verification are built on images extracted from Internet, and they might have images presenting partial occlusion due to glasses, hairstyle changes and make-up. Nevertheless, humans can obtain enough information for recognition purposes from small portions of faces.

The objective of this thesis is to analyze the performance of deep CNN architectures for face-verification using face-images under different occlusion conditions, utilizing different similarity measurements, changing the CNN's architecture or CNN's feature maps, and validating testing procedures, commonly, used in object recognition and verification.

1.1 STRUCTURE

The thesis covers in the chapter 2 the fundamentals of neural networks, giving a small introduction of the biological neuron, its mathematical model, the topology of connections among neurons (the network), explaining the algorithm of how neural networks obtain information automatically from data, and, finally, their application in vision. In the chapter 3, this thesis presents the state of the art in face and object recognition, describing the basis algorithm that this thesis studies. A discussion of an specific deep convolutional neural network architecture for face recognition and verification is described, and the method followed by this thesis is presented in the chapter 4. Chapter 5 shows the experiments and an analysis of their results. In the end, the final conclusion is given in chapter 6.

2 FUNDAMENTALS

2.1 ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANN) aspire to model the information processing capabilities of nervous systems of animals, biological neural networks (BNN) [Roj96] [Kon05], since BNNs have the elementary components of a computing model: storage, transmission, and processing. In general, neurons collect signals from the environment, and they transform and process these signals to produce and transmit an adequate response [Hai14]. Besides, the capability of learning makes them very attractive as a computational intelligence technique [Kon05]

2.1.1 *Biological Neural Networks*

Millions of interconnected cells, called neurons, compose the human nervous system. Each of the neurons is a complex unit that deals with incoming signals, and produces responses. Fig 2.1.1 shows the general structure of a neuron. Neurons are conformed by four elements: dendrites, synapses, a cell body and axons. The dendrites are branches that represent transmission wires for gathering information. They receive signals at contact regions, termed synapses, coming from other neurons. In the cell body, organelles take responsibility for the protein synthesis and metabolism maintenance of the neuron [Hai14]—for example, mitochondrias supply energy—, and the processing of information takes place producing an output signal. This output can be an excitation or an inhibition [Roj96], depending on the type of receptors in the synapses [Hai14]. In the end, the output signal is transmitted via the axon [Roj96].

The neural information processing is a co-action between electrical transmission of information in the cell and the chemical transmission in the synapses [Roj96]. The electrical transmission of information in the cell is produced and transmitted at the cell membrane. The signals are represented by depolarization waves, called action potentials, in which the internal potential of the cell with respect to the exterior of the cell increases from its equilibrium potential to a maximal positive potential, and then, it returns back to its equilibrium state. During the wave, ions of sodium Na^+ flow into the cell, depolarizing it, and then after a short period of time, ions of potassium

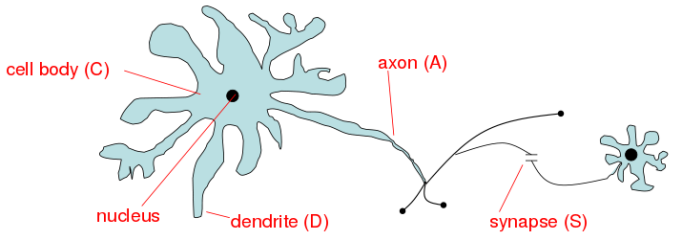


Figure 2.1.1: General structure of a neuron [Roj96]

K^+ flow outwards compensating the depolarization or repolarizing the cell. The cell membrane controls the inwards and outwards of ions, depending on the permeability for each of the ions. The permeability is determined by the quantity and size of pores in the cell membrane. Those pores, called ionic channels, have different forms and charges that allow only specific ions to go in or out [Roj96].

The chemical transmission among cells is a combination of electrical and chemical processes in the neurons, mostly in the synapses. Synapses are conceived as the thickening of axons[Roj96]; essentially, they represent the zone where the axons communicate with other neurons [Hai14]. Synapses consist of three elements: A pre-, inter-, and post- synaptic elements [Hai14]. The pre-synaptic element or pre-synaptic terminal has mitochondrias, which provide energy to the neuron, and small vesicles that contain chemical transmitters. The inter-synaptic element or synaptic gap is the small space between a pre-synaptic element and an attached cell. In the attached cell, the post-synaptic element or post-synaptic membrane is located [Hai14]. When an action potential arrives to the synapse, the pre-synaptic terminal gets depolarized and the vesicles fuse with the pre-synaptic membrane, which liberate their chemical transmitters [Hai14]. The chemical transmitters flow into the synaptic gap, hooking up to the ionic channels of the post-synaptic membrane [Roj96]. Depending on the kind of the chemical transmitters, the ionic channels are opened, and allow chemical transmitters to flow inwards the cell's membrane [Roj96]; this produces an increment or decrement of the membrane potential or, essentially, the membrane depolarizes or hyperpolarizes. This de-, hyper- polarization stimulates the inward-flow of more chemical transmitters, and therefore, the polarization increases or decreases even more— a kind of positive feedback [Hai14]. As a consequence, they alter the cell's potential. If chemical transmitters , mostly ions of sodium, are positive, cell's potential will increase and will assist the neuron to cause an action potential, and hence the neuron will get excited. On the contrary, if the chemical transmitters, mostly ions of

chloride, are negative, cell's potential will decrease, and will hinder the emitting of an action potential; thus, the synapse will behave as inhibitory synapse [Roj96].

For firing an action potential, the depolarization must rise above a threshold potential; that is, enough sodium-selective ionic channels must be opened [Roj96]. When few sodium-selective ionic channels open, the sodium ions start flowing inwards the post-synaptic membrane, and this membrane becomes depolarized. The threshold potential comes along when the total inward-flow of sodium ions is greater than the outward-flow of potassium ions. In that point, the remaining sodium-selective ionic channels are opened, rising quickly the inward-flow of sodium ions, and increasing still more the depolarization. When the depolarization reaches a maximum, the sodium-selective ionic channels close, and the outward-flow of potassium ions repolarizes the membrane potential [Hai14]. On the other hand, if chloride- or potassium-selective ionic channels open, negative ions (chloride ions) will flow inwards, or positive ions (potassium ions) will flow outwards the post-synaptic membrane, and the membrane potential will decrease moving away from the threshold potential [Hai14]; the probability for firing an action potential decreases [Roj96].

The rising above the threshold potential, and consequently the firing of an action potential, could be handled as a combined process among different dendrites. The dendrites collect action potentials coming from different neurons; that means, chemical transmitters from pre-synaptic terminals, coming from other neurons, flow into the post-synaptic membrane—This collection is known as a spatial summation [Roj96]. In the same way, there exists a temporal summation as the frequency of action potentials increases, in which case the membrane potentials are also summed, becoming more effective than unique post-synaptic potentials [Hai14]. These combined summations control the electric activity in the cell's body to determine whether the axon fires an action potential or does not [Hai14]. The quantity of synapses, which are necessary to fire an action potential, is determined by the number of ionic channels per synapse opened by pre-synaptic signals—A single synapse or, simultaneously, various synapses can generate an activation potential in an attached neuron. The number of ionic channels per synapse represent the synapse's efficiencies.

The synapse's efficiencies can be altered by opening more ionic channels, a sort of a membrane's permeability modification; in fact, one of the mechanism that neurons use to learn or store information is altering the synapse's efficiencies [Roj96] — Neurons, also, learn by setting new connections among neurons. Neurons use, for example, NMDA receptors (N-methyl-D-aspartate) as ionic channels that are permeable to different molecules for increasing the depolarization. Initially, these NMDA receptors are blocked by magnesium ions. When an action potential arrives to a synapse, the NMDA receptors lose their magnesium ions, and they become permeable to calcium

or sodium ions; in that way, the threshold potential is altered. Therefore, a cell can be trained to fire easier if a lower threshold potential is set by modifying its membrane's permeability [Hai14]. The synapse's efficiencies are increased when two cells, which are connected by a synapse, fire. Otherwise, the synapse's efficiencies are increased [Kon05].

In the next subsection, the four elements : dendrites, synapses, cell bodies, and axons are adopted as the minimal structure to model artificial neural networks.

2.1.2 *Mathematical representation of a neuron*

Mathematically, a neuron can be seen as a unit that activates depending on its inputs and internal parameters. A neuron has signals that travel along the axons of previous neurons as its input and are transmitted by synapses [Roj96]. The internal parameters represent the synapse's efficiencies of the BNNs (see subsection 2.1.1), which are called weights [Roj96]. The input signals interact by means of a multiplication with the weights. In essence, the idea is that these weights are learnable so they can influence other neurons. This influence can be excitatory with positive-, or inhibitory with negative-weights [Roj96]. For a cell with n synapses, a synapse i is associated with a weight w_i. In the cell body, the input signals, are summed, as the spatial summation of the BNNs does (see subsection 2.1.1). For example, if all synapses are activated at the same time, the total input will be $w_1 + w_2 + ... + w_n$. If the final summation is above a threshold value, then the neuron will activate and send a signal along its axon. This activation is performed by an arbitrary function. In general, a neuron performs a scalar product between its input and its weights, it adds a bias— a negative threshold—, and applies a non-linearity activation function $f(\cdot)$ [FFL16], see Figure 2.1.2.

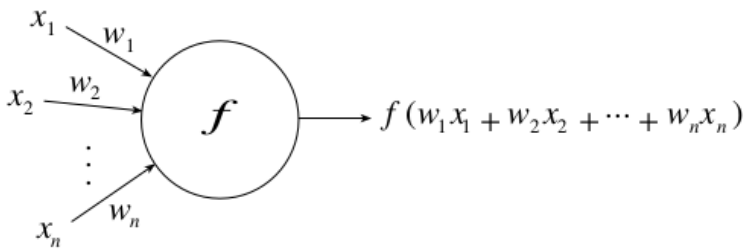


Figure 2.1.2: Mathematical representation of a neuron [Roj96]

An artificial neuron can be considered as a primitive function that is capable of transforming its input into a defined output. A neuron is, in general, a primitive

function of n-arguments producing a numerical output. This primitive function is split into two components: an integration- or linear excitatory/inhibitory- component, and an activation function- or non-linear- component [Kono5]. First, the n-arguments or signals are reduced to single numerical value, a total excitation, by means of a integration function $g(\cdot)$, a summation. Second, the single value is used by the activation function $\varphi(\cdot)$ (see subsection 2.1.5) to produce the final output of the computing unit [Roj96] within a finite range: $\varphi[g(\cdot)]$ [Kono5].

The simplest neuron or unit is the McCulloch-Pitts unit (MCP). It receives and produces only binary signals, $[0, 1]$. Fundamentally, it is a logical or boolean mapping $\Phi : \mathbb{B}^n \to \mathbb{B}$ (see Equation 2.1.2) [Roj96]. A MCP seems very limited, but it contains all necessary features to implement any given logical function of n-arguments, and to implement automaton [Roj96]. A MCP has n-excitatory signals $x_1 + x_2 + ... + x_n$, m-inhibitory signals $y_1 + y_2 + ... + y_m$, all its weights are equal to "1", and its activation function is the step function [Kono5], a comparison between the summed inputs and its threshold, as the BNNs do when a total depolarization rises above a threshold potential (subsection 2.1.1). If at least one of the inhibitory signals is "1", the unit will be inhibited, see Equation 2.1.1. Otherwise, the total excitation is computed and compared with a threshold θ, following the Equation 2.1.2 [Roj96].

$$\bar{\Phi}(x_1, x_2, ..., x_n; y_1, y_2, ..., y_m) = \Phi(x_1, x_2, ..., x_n) \prod_{j+1}^{m} (1 - y_j) \tag{2.1.1}$$

$$\Phi(x_1, x_2, ..., x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i \geqslant \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.1.2}$$

For example, the **OR**, the **AND** and the **NOT** functions can be implemented with MCPs. For the **OR** function, the MCP has a threshold $\theta = 1$; that is, if there is at least one input that is excited, the neuron will fire. In the case of an **AND** function, a n-input MCP has a threshold equal to the number of inputs, threshold $\theta = n$ [Roj96]; that means, if all the n-inputs are excited, the neuron will fire. The MCP computing the **NOT** function has one input signal that is, at the same time, its inhibition signal, and a threshold $\theta = 0$. If a NOT-MCP gets a "1" as input, the output will be "0" due to the inhibition, see Equation 2.1.1. On the other hand, if the input is "0", the output is set to "1" following the Equation 2.1.2.

As any logical function is a combination of sums, products and complements by means of the Boolean algebra—all logical functions can be expressed in two canonical

forms: sum of products (when several product-terms are summed) or product of sums (when several sum-terms are multiplied)[Flooo]— and as the **OR**, **AND**, and **NOT** logic gates represent, respectively, the summation, the product and the complement operations [Flooo]. Any logical functions is thus a combination of **OR**, **AND**, and **NOT** gates. The canonical forms are defined in two steps; first, one computes the terms (sums or products), and second, one computes the final result by summing or multiplying the terms. Since these gates can be implemented with MCP units, it follows that any logical function $\Phi : \mathbb{B}^n \to \mathbb{B}$ can be computed with a MCP network, a connection of multiple artificial neurons of MCP units, of two layers [Roj96] depending on the canonical expression of the logical function.

The MCP networks are called unweighted networks because they use weights equal to "1". However, they can model more general networks with relative excitation and inhibition properties; that is, relative fixed weights [Roj96]. For example, the Figure 2.1.3 shows a fixed weighted network, which has the primitive function $0.2x_1 + 0.4x_2 + 0.3x_3 \geqslant 0.7$. The primitive function is equivalent to $2x_1 + 4x_2 + 3x_3 \geqslant 7$ if one multiplies by 10 both sides of the function. This primitive function can be computed by a MCP network with redundant inputs, see Figure 2.1.3 [Roj96].

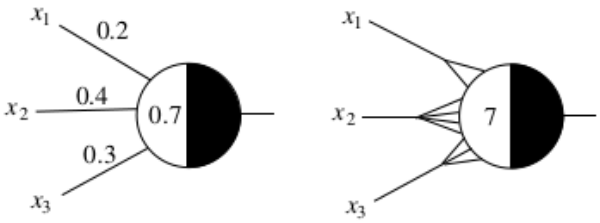


Figure 2.1.3: Equivalent weighted and unweighted networks [Roj96]

The MCP networks and the fixed weighted networks are logic gates. They are used, for example, to simulate any finite automaton [Roj96]. However, their topologies and parameters must be fully specified for any different application. Furthermore, there are no good algorithms for adapting the parameters and the network topology for aiming different problems, and their construction is difficult [Roj96]. Nevertheless, the relative parameters of the fixed weighted networks are relevant. By augmenting the definition of a MCP neuron with real input signals and non-fixed relative parameters, Minsky and Papert conceived the simple perceptron, called Minsky-Papert perceptron (MPP), see Equation 2.1.3 [Kono5]. The main difference between the MCP neuron and the MPP is that the input signals are a subset of $\mathbb{R}$ ($x \in [0, 1] \subset \mathbb{R}$) and the weights are not fixed. Being non fixed weights, one implies that weights can be adapted to solve

different problems. The adjustment of network parameters is called "learning" [Roj96] [Kon05].

$$\Phi(x_1, x_2, ..., x_n) = \begin{cases} 1 & \text{if} \quad \sum_{i=1}^{n} \omega_i x_i \geqslant \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.1.3}$$

A single MPP is a linear classifier— one can see it, also, as a mapping ($\Phi : \mathbb{R}^n \to \mathbb{B}$). It separates the input space in two subspaces whether the result of the primitive function is "1" or "0", see (Equation 2.1.3). For example, the MPP computing the function $0.9x_1 + 0.8x_2 \geqslant 0.6$ divides the 2-D space $(x_1, x_2) \in \mathbb{R}^2$ in two half-planes. By isolating x_2, one gets $x_2 \geqslant \frac{3}{4} - \frac{9}{8}x_1$. All the values above the line $x_2 = \frac{3}{4} - \frac{9}{8}x_1$ satisfy the function Φ, so the MPP output is "1". On the other hand, for the values below the line, the MPP's output is "0" [Kon05].

One can see the threshold θ as a bias, $\sum_{i=1}^{n} \omega_i x_i - \theta \geqslant 0$. In fact, one can extend the input vector $[x_1, x_2, ..., x_n]$ and the weight vector $[\omega_1, \omega_2, ..., \omega_n]$ of the Equation 2.1.3 such that the threshold is converted to a weight [Kon05]. By extending the vectors, one can consider the threshold or bias as a weight.

$$\Phi(x) = \begin{cases} 1 & \text{if} \quad \omega' x \geqslant 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1.4}$$

with $x = x_1, x_2, ..., x_n, 1$, and $\omega = (w_1, w_2, ..., w_n, w_{n+1})$, and $w_{n+1} = -\theta$.

The weights represent the synapse's efficiencies of BNNs, see subsection 2.1.1. As the BNNs adapt their synapse's efficiencies by using NMDA receptors, the MPPs adapt their weights by means of an learning algorithm. The learning algorithm changes the weights using a set of positive (P) and negative (N) samples. It starts setting initial weights at random, and it iterates until the sample's outputs are all correct. For adapting or learning the weights of a MPP, one follows:

1. Choose the extended weight vector ω_0 at random.
2. Choose an arbitrary extended input vector $x \in P \cup N$, with P being a set of positive samples; that is, x that produces an output $= 1$, and N being a set of negative samples; that is, x that produces an output $= 0$.
3. Feed x to the MPP.

4. If output of MPP is wrong, then change the weights.
   if $x \in P$ and $\omega_t' x \leqslant 0$ then $\omega_{t+1} = \omega_t + x$ and $t++$.
   if $x \in N$ and $\omega_t' x \geqslant 0$ then $\omega_{t+1} = \omega_t - x$ and $t++$.
   go to 2.

5. If all inputs vectors and outputs are correct, then stop.

In the end, the MPP will separate the input vectors in two classes with weights describing a line. The weights are changed whenever an extended vector in P or N is not well classified. The line, essentially, is rotated and translated until it separates the inputs in the two classes. This learning algorithm adjusts the network's parameters (weights) in an optimal manner such that the network reflects the known information (inputs) and extrapolates to new patterns [Roj96].

When one has convex sets (non-linearly separable regions), one can use many MPPs in three layers. In the first layers, each of the MPPs computes a linear separation of the sets (subsets with output "1" or "0")— The MPPs map the input vector to a feature space[Roj96]. The second layer is an **AND** MCP for decoding the output of the MPPs [Kono5], which computes an output value; for example, if MCP outpus are all "1" for each region. The last layer is an **OR** MCP that joins the sets.

2.1.3 *Layered-Perceptron and Learning*

The connection of multiple artificial neurons results in an artificial neural network (ANN) [Kono5]. ANNs are, in essence, networks of primitive functions. The author in [Roj96] defines that an ANN, with k neurons, n-dimensional input vector x and m-dimensional output vector y, is a network function ϕ evaluated at the point $x_1 + x_2 + ... + x_n$. The k neurons implement the primitive functions $f_1 + f_2 + ... + f_k$, which are use to produce m-dimensional output vector $y_1 + y_2 + ... + y_m$. One can say that ANN thus behaves as a "mapping machine", capable of modeling a function $\Phi : \mathbb{R}^n \to \mathbb{R}^m$. The ANN, specifically, does not compute the function Φ, but it approaches the function Φ by learning its weights ω based on a training set and a minimization of an objective function $E_p(\omega)$. A training set is a set of sample-tuples (n-dimensional input vectors x and m-dimensional target vectors t). An objective function $E_p(\omega)$ computes the difference between target vectors t and output vectors y of the ANN in terms of its weights. After learning, the function Φ will interpolate outputs for new unknown input vectors recognizing whether a new input vector is similar to the training set (learned samples) and producing a similar output [Roj96]
The ANNs have different connections or topological configurations; for example, the

feed-forward and the feedback topologies [Kono5]. In the feed-forward topologies, the k neurons are subdivided into l subsets or layers $N_1, N_2, ..., N_l$, called hidden layers. The information flows forwards; that means, nodes in a layer compute some outputs, and those outputs are then fed as inputs to neurons in posterior layers—the output from N_1 goes as input to N_2 and so on [Roj96]. As a result, the computation is well defined, and there is no need of synchronization. The number of weights between two layers is the product between the number of neurons n in the first layer and the number of neurons k in the second layer; that is, the total number of weights between two layers is mn [Roj96]. In the case of feedback topologies, the ANN have cycles, in which the output of some neurons at time t are fed back to the same neurons at time $t + 1$, a feedback [Kono5].

2.1.4 *Single Layer Perceptron*

As a linear classifier, the single-layer perceptron (SLP Figure 2.1.4) is the simplest feedforward neural network. It separates the input space in two subspaces as a MPP (see subsection 2.1.2), but it utilizes non-linearity functions as activation functions $\varphi(\cdot)$ (see subsection 2.1.5) instead of a comparison with a threshold value. Similar to MMPs, a SLP has a $n + 1$-dimensional extended-input vector x, and a $n + 1$-dimensional extended-weight vector ω.

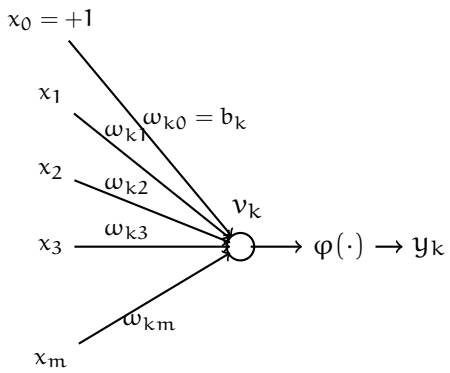


Figure 2.1.4: Single layer perceptron (SLP) [FFL16]

A SLP implements the primitive function 2.1.5.

$$\Phi(x) = \varphi(\omega^T x) \tag{2.1.5}$$

with $x = [x_1, x_2, ..., x_n, 1]^T$ and $\omega = [\omega_1, \omega_2, ..., \omega_n, \omega_{n+1}]$, and $\varphi(\cdot)$ as the activation function.

Firstly, a SLP computes a dot product between the extended-input vector x and the extended-weight vector ω, analogous to the spatial summation of BNNs (see subsection 2.1.1). Secondly, the neuron computes an output by means of a non-linearity as the activation function $\varphi(\cdot)$ (see subsection 2.1.5). For adapting the weights of a SLP, one seeks to minimize an objective function E_p in weight-space. As the objective function E_p is computed with respect to the weights, one wants, specifically, the weights for which the objective function is minimal. For that end, a SLP uses an optimization algorithm, called gradient descend (see subsection 2.1.6).

2.1.5 *Activation Function*

In the initial examples of MCPs and MPPs, one used the step, a comparison between the summed inputs and a threshold (see subsection 2.1.2), as the activation function $\varphi(\cdot)$. However, ANNs utilize non-linearity functions, which take single values and "squash" them within a finite range [Kono5] by performing a mathematical operation. The reason for using activation functions is connected with the optimization algorithm (explained in subsection 2.1.6) for adapting the weights. In essence, the optimization algorithm requires computations of gradients, and non-linerity functions are continuous and differentiable [Roj96].

The main non-linearity functions and their derivatives (important for the optimization algorithm, see subsection 2.1.6) are [FFL16].

- The sigmoid function: It takes a value and forces it to the range $[0, 1]$ with interception with y-axis in 0.5. For large negative values, the output becomes "0", and for large positive values, the output becomes "1".

$$\varphi(x) = \text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.1.6}$$

$$\varphi'(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \varphi(x)\,(1 - \varphi(x)) \tag{2.1.7}$$

- The tanh function: It takes a value and forces it to the range $[-1, 1]$. Unlike the sigmoid, the tanh is zero-centered [FFL16]; in essence, a tanh is a scaled and translated sigmoid function [Roj96].

$$\varphi(x) = \tanh(x) = 2(\text{sigmoid}(x) - 1) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{2.1.8}$$

$$\varphi'(x) = \tanh'(x) = (1 - \tanh^2(x)) \tag{2.1.9}$$

- The ReLU: The Rectified Linear Unit thresholds at zero Figure 2.1.5, proposed by the authors in [GBB11]. It achieves better performance than the saturating ones, because the activation function's response is closer to the biological neuron's one, the action potentials see subsection 2.1.1, obtaining sparse output representations [GBB11]. Besides, using the ReLU is cheaper computationally, since only a subset of neurons remains active. ReLu controls the number of active neurons, which become a path between the output and the input; that is, the relation between the output of activated neurons is linear with respect to the input. This helps, also, the optimization algorithm, since gradients only flow back on the path of activated neurons and the remaining ones stay unchanged, see subsection 2.1.6 [GBB11], which supports the conclusion of the authors in [AK12] showing that the ReLU is faster for training than the saturating non-linearities (Sigmoid and Tanh).

$$\varphi(x) = \max(0, x) \tag{2.1.10}$$

$$\varphi'(x) = \begin{cases} 0 & \text{if} \quad x \leqslant 0 \\ 1 & \text{if} \quad x > 0 \end{cases} \tag{2.1.11}$$

- The Leaky ReLU. The function computes a small negative slope α instead of thresholding to zero as ReLUs.

$$\varphi(x) = \begin{cases} \alpha x & \text{if} \quad x < 0 \\ x & \text{if} \quad x \geqslant 0 \end{cases} \tag{2.1.12}$$

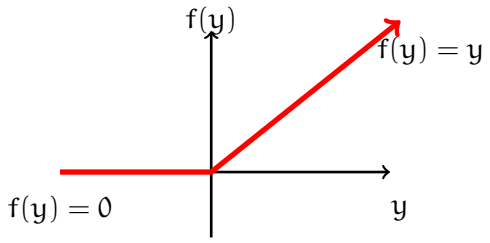


Figure 2.1.5: Rectified Linear Unit [AK12]

$$\varphi'(x) = \begin{cases} \alpha & \text{if} \quad x \leqslant 0 \\ 1 & \text{if} \quad x > 0 \end{cases} \tag{2.1.13}$$

2.1.6 *Gradient Descent (GD)*

GD is an optimization algorithm that finds a local minimum of an objective function by taking steps proportional to the negative gradient of the function. For ANNs, It repeatedly computes the gradient of an objective function $E_p(\omega)$ with respect to the weights, and it updates the network's parameters or weights, see equation 2.1.14, [FFL16].

$$\omega_{i+1} = \omega_i - \gamma \Delta E_p(\omega) \tag{2.1.14}$$

with ω_i the weights of a ANN in step i, ω_{i+1} the updated weights, γ a constant parameter, $E_p(\omega)$ the objective function, and $\Delta E_p(\omega)$ equal to.

$$\Delta E_p(\omega) = \frac{\partial E_p(\omega)}{\partial \omega} = \left[\frac{\partial E_p(\omega)}{\partial \omega_1}, \frac{\partial E_p(\omega)}{\partial \omega_2}, ..., \frac{\partial E_p(\omega)}{\partial \omega_k} \right] \tag{2.1.15}$$

for a k-dimensional weight vector ω

The parameter γ is the step length in the negative gradient direction, called the learning rate [Roj96]. It is an important parameter since small steps lead to the expected direction but slowly, and large steps progress faster but it might overshoot and produce bigger values of $E_p(\omega)$ [FFL16]. The objective function $E_p(\omega)$ computes the difference between the target vectors t and the computed output vectors y of an

ANN. It can be e.g. the total sum squared error (equation 2.1.16) [Roj96] [LDS+89] or the empirical softmax log-loss (equation 2.1.17) [PVZ15].

$$E_p(\omega) = \sum_{i=(x,t)\in D} \frac{1}{2} \sum_{m=1}^{M} (t_{m_i} - y_{m_i})^2 \tag{2.1.16}$$

$$E_p(\omega) == - \sum_{i=(x,t)\in D} \left(\frac{e^{\langle t_i, x_i \rangle}}{\sum_{q=(x,t)\in D} e^{\langle t_q, x_i \rangle}} \right) \tag{2.1.17}$$

The objective function $E_p(\omega)$ (Equation 2.1.16 or Equation 2.1.17) is considered with respect to a training set D (sample-tuples (x, t)). There are three versions of GD depending on the set D of sample-tuples: It is called simply **GD** when it takes the whole sample-tuple set as D. If the GD takes just one random sample-tuple as D updating the weights sample-tuple by sample-tuple, the GD will be called **Stochastic Gradient Descend**. If the GD takes a batch or subset of the sample-tuple set as D updating the weights just after feeding the batch to the network then the GD will be called **Batch Gradient Descend**.

There are, also, other versions of the GD, which add more parameters. For example, the GD with momentum takes into account previous changes of the weights, see Equation 2.1.18,

$$\begin{aligned} \nu_{i+1} &= \gamma_1 \nu_i - \gamma_2 \Delta E_p(\omega) \\ \omega_{i+1} &= \omega_i + \nu_{i+1} \end{aligned} \tag{2.1.18}$$

Or the GD with momentum and weight decay, see Equation 2.1.19. This GD is used frequently in the networks that this thesis focused.

$$\begin{aligned} \nu_{i+1} &= \gamma_1 \nu_i - \gamma_2 \Delta E_p(\omega) - \gamma_3 \gamma_2 \omega_i \\ \omega_{i+1} &= \omega_i + \nu_{i+1} \end{aligned} \tag{2.1.19}$$

with ω_i the weights of a ANN in step i, ω_{i+1} the updated weights, ν_i the weight changes in step i, ν_{i+1} the updated weight changes, γ_1 the momentum constant, γ_2 the learning rate, γ_3 the weight decay, and $E_p(\omega)$ the objective function.

Example

For a SLP (see subsection 2.1.4) and the total squared error (Equation 2.1.16) as the objective function $E_p(\omega)$ with $M = 1$, one has:

$$E_p(\omega) = \sum_{i=(x,t)\in D} \frac{1}{2}(t_i - \varphi(\omega_i^T x_i))^2 \tag{2.1.20}$$

The computation of the gradient $\Delta E_p(\omega)$ for a set D of sample-tuples (x, t) is:

$$\frac{\partial E_p(\omega)}{\partial \omega} = \frac{\partial}{\partial \omega}\left(\sum_{i=(x,t)\in D} \frac{1}{2}(t_i - \varphi(\omega_i^T x_i))^2\right) \tag{2.1.21}$$

$$\frac{\partial E_p(\omega)}{\partial \omega} = -\sum_{i=(x,t)\in D} (t_i - \varphi(\omega_i^T x_i))\varphi'(\omega_i^T x_i)x_i \tag{2.1.22}$$

In the Equation 2.1.22, one notices that the gradient of $E_p(\omega)$ depends on the derivative of the activation function or non-linearity $\varphi'(\cdot)$, see subsection 2.1.5. Thus, the GD behaves different for each of the non-linearity functions. For example, the sigmoid saturates at "0" or "1", and its derivative is almost "0". If the gradient is very small or almost "0", the weights will not change causing that the ANN will not learn [FFL16].

On the assumption that one takes the derivative of the sigmoid (Equation 2.1.7) as $\varphi'(\cdot)$, the final gradient $\Delta E_p(\omega)$, from Equation 2.1.22, is:

$$\frac{\partial E_p(\omega)}{\partial \omega} = -\sum_{i=(x,t)\in D} (t_i - y_{i_{output}})y_{i_{output}}(1 - y_{i_{output}})x_i \tag{2.1.23}$$

Multilayer Perceptron

A multilayer perceptron (MLP) is an ANN with multiple hidden layers. Its layers are fully-connected layers since neurons between two layers are completely connected from each other [FFL16]. The Figure 2.1.6 shows an example of a MLP of 3 layers: n neurons in the input layer, k neurons in the hidden layer, and m neurons in the output layer. This MLP has a n-dimensional input vector x, a m-dimensional output vector y, and a k-dimensional vector z for the output of the hidden neurons. Therefore, it has a

kn-dimensional matrix W_1 as weights between the hidden layer and the input layer, and a mk-dimensional matrix W_2 as weights between the output layer and the hidden layer.

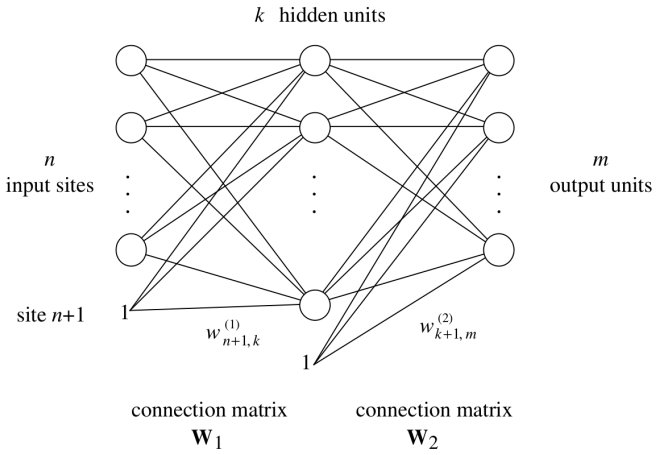


Figure 2.1.6: Multi layer perceptron MLP [Roj96]

The MLP in the figure (2.1.6) implements the mapping $\Phi : \mathbb{R}^n \to \mathbb{R}^m$ as follows:

$$z = \varphi(W_1 \cdot x) \tag{2.1.24}$$

$$y = \varphi(W_2 \cdot z) = \varphi(W_2 \cdot \varphi(W_1 \cdot x)) \tag{2.1.25}$$

with $x = [x_1, x_2, ..., x_n, 1]^T$, $z = [z_1, z_2, ..., z_k, 1]^T$, and $y = [y_1, y_2, ..., y_m]^T$.

One notices in the Equation 2.1.25 that the output y is a function composition since the hidden neuron's output becomes the input of the last layer neurons. In general, a MLP represents a chain of function compositions, where output of layers become inputs of following layers [Roj96], implementing the function Φ. This composition is important if one wants to update the weights of the MLP for implementing the function Φ following a training set, because one can not use just the GD (see subsection 2.1.6) since the hidden neurons do not have target values, as the neurons in the last layer do (see subsection 2.1.4). However, one can use the chain rule for differentiating a function composition, and one can implement the GD following the Backpropagation algorithm (see subsection 2.1.7).

2.1.7 *BackPropagation*

The Backpropagation algorithm uses the GD (subsection 2.1.6) for finding a combination of weights of a MLP that minimize an objective function [Roj96] such as the MLP implements the mapping $\Phi : \mathbb{R}^n \to \mathbb{R}^m$. The Backpropagation is divided in two phases: A forward step and a backward step. In the forward step, sample-tuples (x, t) are fed to the MLP. The hidden neurons k in the MLP compute their outputs layer by layer until a final output y is produced. Then, the backward step computes the gradients with respect to the weights from the final layer to the initial layer—The gradients are propagated backwards by using successively the chain rule [FFL16]—and it, finally, updates the weights by taking a fraction of the computed gradient.

Example

For better describing the Backpropagation, one takes the MLP in the Figure 2.1.6, the total sum squared error (Equation 2.1.16) as the objective function $E_p(\omega)$ and a training set D (sample-tuples (x, t)). First, the MLP sends sample-tuples (x, t) forwards computing a k-dimensional hidden vector z, and a m-dimensional output vector y. The relations among the three vectors are described in the equations (2.1.24) and (2.1.25). Second, the Backpropagation implements the backward step; that is, it computes the gradients for each layer, and it updates the weights following the updating rule of GD (see Equation 2.1.14). The weight-updates for the MLP are:

$$W_{2_{t+1}} = W_{2_t} - \gamma \frac{\partial E_p(W_1, W_2)}{\partial W_2} \tag{2.1.26}$$

$$W_{1_{t+1}} = W_{1_t} - \gamma \frac{\partial E_p(W_1, W_2)}{\partial W_1} \tag{2.1.27}$$

being W_1 a kn-dimensional matrix, and W_2 a mk-dimensional matrix.

Each of the weight-update rules has a gradient of the objective function with respect to each of the weights, in this case all the terms in the matrices W_1 and W_2. Rewriting the objective function $E_p(\omega)$ (see Equation 2.1.16) in terms of W_1 and W_2, one has:

$$E_p(W_1, W_2) = \sum_{i=(x,t)\in D} \frac{1}{2} \sum_{m=1}^{M} [t_{m_i} - \varphi_m(W_2 \cdot z_i)]^2 \tag{2.1.28}$$

$$E_p(W_1, W_2) = \sum_{i=(x,t)\in D} \frac{1}{2} \sum_{m=1}^{M} [t_{m_i} - \varphi_m(W_2 \cdot \varphi(W_1 \cdot x_i))]^2 \tag{2.1.29}$$

Based on that, one can see that each of the final neurons adds its quadratic deviation to the objective function $E_p(W_1, W_2)$. If we consider each of the quadratic deviations separately, the gradient of the objective function Equation 2.1.28 with respect to each of the weights in the matrix W_2 will be:

$$\frac{\partial E_p(W_1, W_2)}{\partial W_2} = - \sum_{i=(x,t)\in D} (t_i - y_i)\varphi'(W_2 \cdot z_i)z_i \tag{2.1.30}$$

And the gradient of the Equation 2.1.29 with respect to W_1 is:

$$\frac{\partial E_p(W_1, W_2)}{\partial W_1} = - \sum_{i=(x,t)\in D} \sum_{m=1}^{M} [(t_i - y_i)\varphi'(W_2 \cdot z_i)W_2\varphi'(W_1 \cdot x_i)x_i]_m \tag{2.1.31}$$

Rewriting the Equation 2.1.31 to Equation 2.1.32, one notice that the factors multiplying the vector z_i in the Equation 2.1.30 are also present in Equation 2.1.32. This factors are called local gradients or error signals of layer inputs with respect to layers outputs [FFL16]. For example, the local gradient of the output-layer is δ_y.

$$\frac{\partial E_p(W_1, W_2)}{\partial W_1} = - \sum_{i=(x,t)\in D} \sum_{m=1}^{M} [\underbrace{(t_i - y_i)\varphi'(W_2 \cdot z_i)}_{\delta_y} W_2\varphi'(W_1 \cdot x_i)x_i]_m \tag{2.1.32}$$

The equations (2.1.30 and 2.1.32) are described with respect to local gradients:

$$\frac{\partial E_p(W_1, W_2)}{\partial W_2} = - \sum_{i=(x,t)\in D} \delta_{y_i} z_i \tag{2.1.33}$$

$$\frac{\partial E_p(W_1, W_2)}{\partial W_1} = - \sum_{i=(x,t)\in D} \sum_{m=1}^{M} [\delta_{y_{m_i}} W_{2_{mk}}]\varphi'(W_1 \cdot x_i)x_i = - \sum_{i=(x,t)\in D} \delta_{z_i} x_i \tag{2.1.34}$$

with,

$$\delta_{y_i} = (t_i - y_i)\varphi'(W_2 \cdot z_i) \tag{2.1.35}$$

$$\delta_{z_i} = \sum_{m=1}^{M} [\delta_{y_{m_i}} W_{2_{mk}}]\varphi'(W_1 \cdot x_i) \tag{2.1.36}$$

The local gradients are computed by using the derivatives of the activation functions (see subsection 2.1.5). Since the derivatives of the activation functions are expressed in terms of their outputs (the neurons outputs), the local gradients are also expressed in terms of the neurons outputs. Thus, the order of the Backpropagation is relevant: First, the neurons outputs are computed, and then the gradients. In fact, the Backpropagation propagates backwards the local gradients or error signals—The gradient of a hidden layer depends on the gradient of the next layer.

Example local gradients

If one uses the sigmoid as an activation function (Equation 2.1.6), one will have the same expression as Equation 2.1.23 for each of the neurons. By organizing the expressions in matrices, the local gradients will become [Roj96]:

$$\delta_{y_i} = diag(y_i)[I_{mxm} - diag(y_i)](t_i - y_i)' \tag{2.1.37}$$

$$\delta_{z_i} = diag(z_i)[I_{kxk} - diag(z_i)]W_2'\delta_{y_i} \tag{2.1.38}$$

with,

$$diag_{ij}(v) = \begin{cases} v_i & i = j \\ 0 & otherwise \end{cases} \tag{2.1.39}$$

And the weight-updates

$$W_{2_{t+1}} = W_{2_t} - \gamma\delta_{y_i} z_i \tag{2.1.40}$$

$$W_{1_{t+1}} = W_{1_t} - \gamma \delta_{z_i} x_i \tag{2.1.41}$$

In general for a MLP with L layers $(N_1, N_2, ..., N_L)$, weight matrices $W_{l,l+1}$ for $l = 1, 2..., L-1$, the total sum squared error as objective function, and using the sigmoid as activation function, the local gradients are:

$$\delta_l = \begin{cases} diag(y_{l_i})[I_{mxm} - diag(y_{l_i})][t_i - y_{l_i}]' & l = L \\ diag(y_{l_i})[I_{k_l x k_l} - diag(z_{l_i})]W'_{l,l+1}\delta_{l+1} & l = 1, 2, ..., L-1 \end{cases} \tag{2.1.42}$$

And the weight-updates

$$\left\{ W_{l_{t+1}} = W_{l_t} - \gamma \delta_{l_i} z_{l_i} \quad l = 1, 2, ...L \right. \tag{2.1.43}$$

In addition to BackPropagation, variations have been used to enhance the training learning, such as, Backpropagation with momentum, QuickProp, Resilient Propagation, etc. The Backpropagation with momentum takes into account the previous change of weights (see Equation 2.1.44).

$$\omega_{i+1} = \omega_i - \gamma \Delta E_p(\omega) + \alpha \Delta \omega_{i-1} \tag{2.1.44}$$

It helps to avoid oscillations of the gradient direction [Roj96], and to eliminate the problem of trapping at local minima [Kono5]. The parameter α is called momentum rate.

2.1.8 *Convolutional Neural Networks*

Convolutional Neural Networks (CNN), first proposed by [FM82], are reciprocal to ANNs (see subsection 2.1.3); that is, CNNs consist of neurons that have parameters: learnable weights and biases, which are organized in a hierarchical structure (layers). They map their input into a more compact representation, or they classify or distribute their input into classes, depending on their objective function [PVZ15]. However, their structure is partially distinct [AK12]: their neurons are 3D filters, and their layers are organized in different kinds. Neurons in a CNN are 3D filters that activate depending on their inputs. They are connected just to a small region, called receptive field [Roj96],

of previous neuron's activations. They compute a convolution operation [1] between the connected-inputs and their internal parameters, and they get activated depending on their output and a non-linearity function 2.1.5 [FFL16] [FM82].

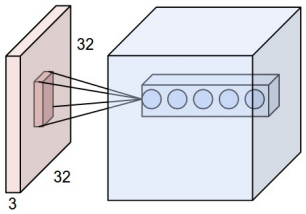


Figure 2.1.7: A neuron in a CNN [FFL16]

The advantage of CNNs over ANNs is that they assume inputs are images (width, height, channel) [LKF+10]. They allow to encode certain properties of images into their architecture; their neurons activate when they "see" particular features extracted at different locations in the image, see Figure 2.1.8; and they are, also, robust against shifts in position and feature distortions [FM82]. CNNs arrange neurons in layers with a 3D shape, see Figure 2.1.7, called convolutional layers. Each of the layers gets a 3D-input volume, called feature-map, and transforms it to another by means of convolutions and a non-linearity [FFL16] [LKF+10]. By stacking layers and downsampling their outputs, CNNs extract more complex and abstract feature-maps, which are, at the same time, invariant to distortions and translations [LCDH+90] [LDS+89]. The last layers of CNNs, see Figure 2.1.8, are standard fully connected layers, same as the ANNs (Equation 2.1.6). These layers compute final descriptors of the input images, which can be considered as global representations of images, or they classify the input images into classes depending on a objective function.

CNNs have less parameters than ANNs because of the neuron's receptive fields and the convolutional operation— since, one slides the same filter through the entire feature-input. Thus, CNNs are easier to train than ANNs [AK12]. To learn the internal parameters or weights of a CNN, one uses the Backpropagation algorithm (see subsection 2.1.7) with Batch Gradient Descend (see subsection 2.1.6) as for the ANNs. In the latest works, authors have used Backpropagation with momentum and weight decay as the learning algorithm, see Equation 2.1.19. The learning algorithm uses a training-image set of sample image-tuples (I, t) composed by images (I) and their respective labels (t). It takes, for example, the Softmax log-loss (Equation 2.1.17) as

1 In image processing, a convolution is the process of multiplying the pixel's and its neighbor's intensities by a kernel or small matrix, which is slided through the entire image along the spatial dimensions.

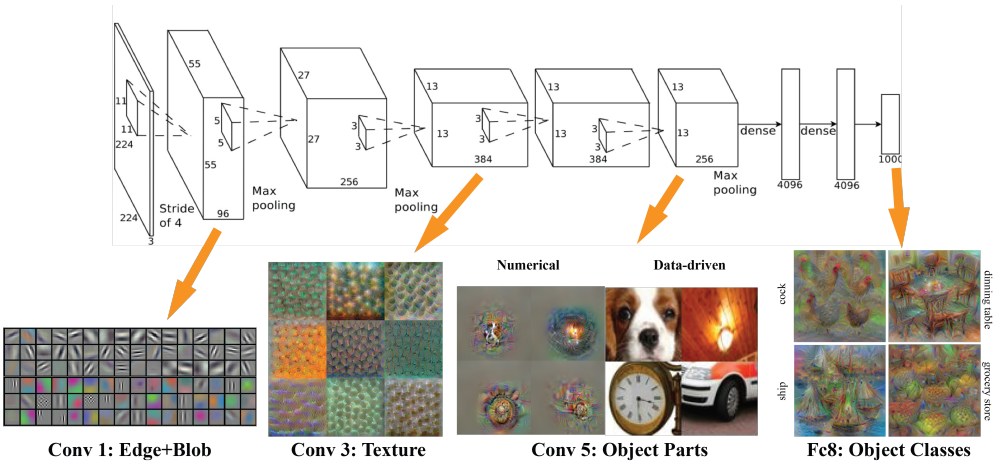


Figure 2.1.8: A Convolutional Neural Network CNN [FFL16]

objective function (see subsection 2.1.6), where the target vector t is the ground-truth class identity [PVZ15].

Architecture

A CNN is a feedforward network composed by layers, as an ANN (2.1.3), that transform an input image from the original pixel values to final class scores by forwarding it layer by layer. Its layers are not all fully-connected, but they are ,in general, of four types:

Input layers

The input is, typically, RGB images (I). One considers them as 3D feature maps of size (Width, Height, Channel).

Convolutional layers

They can be interpreted as holding neurons arranged in a 3D volume. It consists of learnable filters (neurons) that give weights to some specific type of feature at some spatial position in the feature-map input X producing a feature-map of weighted

summations Y. Each of the neurons computes convolutions with small regions in X, see (Equation 2.1.45) [LKF+10].

$$y_j = b_j + \sum_{x_i \in X} w_{ij} * x_i \tag{2.1.45}$$

with $y_j \in Y$, $j = 1, 2, ..., D$, and D the depth of the convolutional layer.

Each filter $w_{i,j}$ is a 3D matrix of size $[F \times F \times C_x]$. Its size is determined by a chosen receptive field (F), see Figure 2.1.7 [FFL16], and its feature-map input's depth (C_x); for example, if the receptive field is 5 pixels and the feature-map input X is a $[32 \times 32 \times 3]$ RGB image, then the filter's size will be $[5, 5, 3]$. One can say that the filter's size represents the number of weights that a neuron has connecting to a region in the input.

The convolutional operation performs similar to the spatial summation of BNNs (subsection 2.1.1), collecting information coming from previous feature-maps. A single convolutional layer produces 2-dimensional weighted feature-maps y_j by convolving each of its filters (see Figure 2.1.9) along the spatial dimensions (width and height) of the feature-map input X—each of the 2-dimensional weighted feature-maps y_j is a weighted summation per filter at every spatial position [Roj96]. As convolutional layers use one filter along the spatial dimensions of the input, one can say that the parameters per filter are shared along those dimensions — detecting the same feature along the height and the width—, what the author in [LDS+89] called weight sharing. The final 3D weighted feature-map output Y of the convolutional layer corresponds to the stacking of the 2-dimensional weighted feature-maps y_j along the depth dimension D of the layer [FFL16] [LDS+89]. The size of the weighted feature-map output $[W_y, H_y, D_y]$ depends on three hyper-parameters of the layer: the depth D, stride S, and zero-padding P. The D parameter represents the number of neurons along the depth dimension of the convolutional layer. The S is the number of pixels that one slides each filter along the width and height of a feature-map input, and P is the number of zeros in the border of a feature-map input to fit neurons along the spatial dimensions of the feature-map [FFL16].

$$W_y = \frac{(W_x - F + 2P)}{S} + 1 \tag{2.1.46}$$

$$H_y = \frac{(H_x - F + 2P)}{S} + 1 \tag{2.1.47}$$

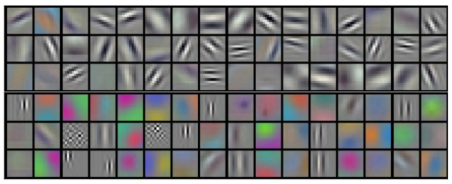


Figure 2.1.9: Filters in a convoluional layer [FFL16]

and $D_y = D$

The number of parameters of a convolutional layer in a CNN in comparison with a fully-connected layer of an ANN for dealing with images is lesser since each of the filters is just connected with a small portion of the feature-map input (given by the receptive field), and the weight sharing along the spatial dimensions of the feature-map input [LDS+89]. For example, if one has a feature-map input of size [32x32x3], and a receptive field of $F = 5$, the number of parameters per neuron in a fully-connected layer will be $(32 * 32 * 3) + 1(\texttt{bias}) = 3073$, and the number of parameters for a neuron (filter) in a convolutional layer will be $(5x5x3) + 1(\texttt{bias}) = 76$ [FFL16]. This reduction of parameters is an advantage of CNNs because their generalisation performance is improved; that is, a learned CNN will not just model the training data, but also, they will generalize new data more accurate than ANNs, in other words, they avoid overfitting, and the learning speed is increased [LDS+89].

By stacking convolutional layers, and downsampling feature-maps, filters in each layer will learn to give more weight and to activate to different patterns: from simple pixels to edglets (oriented-edges or blob of colors), edglets to motifs, and eventually, from motifs to more complex patterns; such as, parts of faces, or parts of cars, see Figure 2.1.10, [FFL16]. Since neurons are connected in a hierarchical structure, the deepest neuron's receptive fields cover bigger areas from the image input. Besides, the deepest neurons activate to combination of features from earlier feature-maps in previous convolutional layers [FM82]. As a consequence, the activation of deepest neurons are not affected by shift variations and feature deformations in the image input [FM82].

Activation layer or Non-linearity Layer

It is a set of activation functions $\varphi(\cdot)$ (subsection 2.1.5) for each of the neurons in the convolutional layer, see Equation 2.1.48. They map a weighted feature-map Y from a convolutional layer to a feature-map of activations Y_{act}, see Equation 2.1.48. Initially, sigmoid (Equation 2.1.6) and tanh (Equation 2.1.8) functions were used as activation

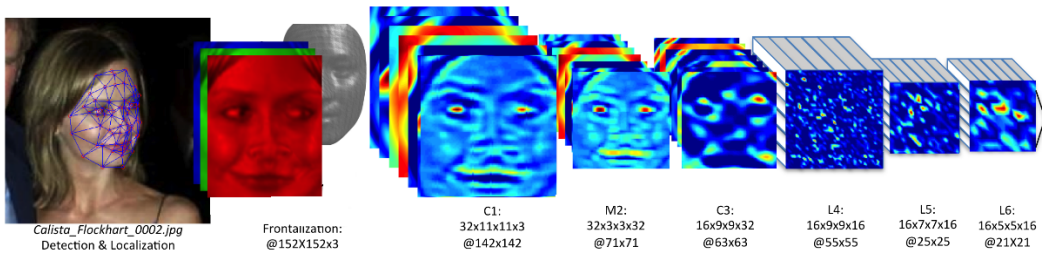


Figure 2.1.10: Feature-maps in a CNN [TYRW14]

functions, but, recently, ReLUs (Equation 2.1.10) have shown better performance for training CNNs [AK12].

$$y_{j_{act}} = \varphi(y_j = b_j + \sum_i w_{ij} * x_i) \tag{2.1.48}$$

with $y_{j_{act}} \in Y_{act}$

Pooling layers

They reduce the spatial size of the feature-maps by downsampling. They are placed among different convolutional layers to reduce the number of parameters in the network [FFL16], and to induce a slightly translation robustness. It takes small squared regions of size F_p with a stride S_p along the spatial dimensions of the feature-map input, and it performs a downsampling operation per region. This operation can be the average over the regions [LKF+10], or the maximum value in each region [FFL16]. The final size of the downsampled feature-map is $[W_p, H_p, D_p]$ for a feature-map input Y_{act}, see Figure 2.1.11.

$$W_p = \frac{(W_y - F)}{S_p} + 1 \tag{2.1.49}$$

$$H_p = \frac{(H_y - F)}{S_p} + 1 \tag{2.1.50}$$

and $D_p = D_y$

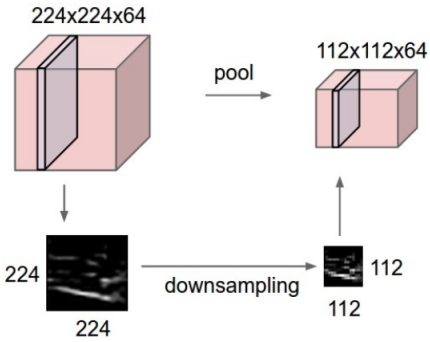


Figure 2.1.11: Pooling layer [FFL16]

Fully-connected layers

These layers have the same topology of a MLP, see Equation 2.1.6, or one can see them as a convolutional layer but their filters are fully connected to a feature-map input [FFL16]. Those layers perform a matrix multiplication with the feature-map input, add a bias, and they use non-linearity function, see subsection 2.1.5, to get activated.

3

RELATED WORK

Face recognition is a specific case of object recognition. Its goal is the identification of unknown face images when a set of known face images is given [BHK97]. However, face recognition is not a simple task, because images of human faces have multiple variations, e.g. different colors, poses, expressions, and sizes, or they can be affected by illumination conditions and occlusion [AR15]. A face recognition model must be invariant to the aforementioned variations, but simultaneously it must be sensitive to variations among faces from different persons (inter-class variations). Ideally, Face recognition aspires to work similar to human perception—humans can recognize a big number of faces through their life, identifying known faces after years of separation, or under extreme occlusion conditions (with beards, glasses, make-up, or even just by looking at small parts of the faces) [TP91]. Today, Face recognition has a great variety of applications such as robot-human interactions, control by gesture, security, people tracking, film industry (enhancement and noise reduction in films); thus, it is an active topic in the computer vision community [AHP06].

This chapter covers, firstly, four face recognition methods that are called, today in the literature "shallow methods". Secondly, two CNNs, see subsection 2.1.8, for vision task that introduce two architectures for object recognition, which are not specifically for face recognition, but they are the basis of the approach that this thesis presents.

3.1 SHALLOW METHODS FOR FACE RECOGNITION

The term "Shallow methods" in face recognition is used for calling all the algorithms that are not based on machine learning—systems that learn automatically from data—, since they extract face-image representations by applying handcrafted descriptors [PVZ15]. Following, four methods are introduced: Landmark based, Eigenfaces, FisherFaces, and LBP.

3.1.1 *Landmark based*

One of the first and the most intuitive approaches of face recognition methods is based on facial landmark detection. The author in [Kan73] built face descriptors

by concatenating geometrical information (area, position in an image, shape, and curvature), manually, extracted from more than 30 feature points, face landmarks, in face-binary images—a face image is binarised by applying a laplacian operator. The face landmarks were: top of the head, cheeks and sides of face, nose, mouth and chin, chin contour, face-side lines, nose lines, eyes, and face axis. Later, the authors in [CEL87] built face descriptors by automatically measuring image coordinates and outlines of four face landmarks: the head outline, lips, eyebrows and eye centers, from gray-scale face-images.

The landmark based algorithms performed the recognition by computing the Euclidean distance between a query image descriptor and the descriptors from a set of training face-images, and by taking the identity of the face-image that corresponds to the shortest Euclidean distance. Nevertheless, they recognize faces in images taken under specific conditions; besides, they required accurate registration of the face landmarks, and they were not notably robust against occlusion, poses, expressions, and illumination since geometrical features might not carry enough information for the task [Kan73].

3.1.2 *EigenFaces*

The Eigenfaces method is a holistic algorithm of face recognition, which considers face images as points in a high-dimensional image space, and it finds a lower-dimensional feature space, called face-space, where recognition becomes easier. The low-dimensional space is spanned by the most significant variations among a set of training face images; that is, the directions with the greatest variance across a set of training face images span the face-space[TP91] [BHK97]. These directions correspond to the eigenvectors of the training set that characterize the variation among all the face images. The eigenvectors and, thus, the face-space are computed by using principal components analysis (PCA) over normalized face images.

For recognizing faces:

1. One projects query face images into the face-space obtaining lower-dimensional representations of the queries.

2. One obtains reconstructions of the queries from the lower-dimensional representations.

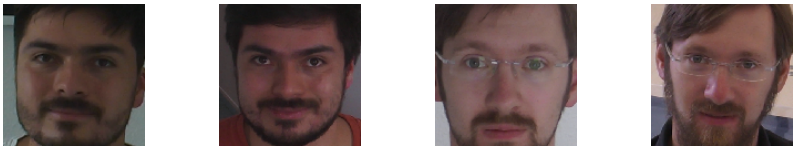
3. One compares the distances among the reconstructed query images, the known reconstructed training images, and the face-space.

By using a lower-dimensional representation, the recognition becomes fast, simple, accurate (under constrained conditions: straight views, maximal 45° face rotation, and indoor images), and robust to small changes in the face images [TP91].

Mathematically, the EigenFaces method finds the eigenvectors of the covariance matrix from a set of training face images, where each face image corresponds to a vector in a high dimensional space [TP91]. The Eigenfaces steps, which were presented by the authors in [TP91], follow:

Training set

Collect a set of training images from M known faces, see Figure 3.1.1.



(a) (b) (c) (d)

Figure 3.1.1: Some samples from a set of two classes

Principal Component Analysis

Compute the principal components of the covariance matrix from the training set.

- Consider a face image $I(x,y) \in \mathbb{R}_{[N\times N]}$ as a vector $\gamma \in \mathbb{R}_{[N^2\times 1]}$.
- Consider the training set as a matrix $\Gamma = \gamma_1, \gamma_2, ..., \gamma_M \in \mathbb{R}_{[N^2\times M]}$, and the average face $\Psi = \frac{1}{M}\sum_{m=1}^{M}\gamma_m$.
- One computes the vectors that best describe the variation of the training set. For that, one obtains the covariance matrix C from the training matrix Γ.

$$C = \frac{1}{M}\sum_{m=1}^{M}[\Gamma_m - \Psi][\Gamma_m - \Psi]^T = \frac{1}{M}\sum_{m=1}^{M}\Phi_m\Phi_n^T = AA^T \tag{3.1.1}$$

with normalized images $\Phi = \Gamma - \Psi \in \mathbb{R}_{[N^2\times 1]}$, and $A = [\Phi_1, \Phi_2, ..., \Phi_m] \in \mathbb{R}_{[N^2\times M]}$.

The needed directions of the lower-dimensional space are, specifically, the eigenvectors Υ from the matrix C, which are given by:

$$C\Upsilon = \Upsilon\Lambda \Rightarrow AA^T\Upsilon = \Upsilon\Lambda \tag{3.1.2}$$

being $\Lambda \in \mathbb{R}_{[N^2 \times N^2]}$ the eigenvalues of C.

As the matrix C is of size $[N^2 \times N^2]$, one can compute N^2 number of eigenvectors Υ and eigenvalues Λ. However, the computation of these eigen- vectors and values is expensive and not necessary, because one has just M number of known images; and M being smaller than N^2 ($M << N^2$), there will be only M meaningful eigenvalues, and the remaining ones will be zero. By taking the matrix $L = A^TA \in \mathbb{R}_{[M \times M]}$, smaller than matrix C, one can find easier the M meaningful eigenvectors Υ of C. This matrix L has a M eigenvectors $\Upsilon_L \in \mathbb{R}_{[M \times M]}$ and eigenvalues $\Lambda_L \in \mathbb{R}_{[M \times M]}$ that are computed by:

$$L\Upsilon_L = \Upsilon_L\Lambda_L \Rightarrow A^TA\Upsilon_L = \Upsilon_L\Lambda_L \tag{3.1.3}$$

The eigenvectors $\Upsilon_L \in \mathbb{R}_{[M \times M]}$ of L and the eigenvectors $\Upsilon \in \mathbb{R}_{[N^2 \times N^2]}$ of C are related. Pre-multiplying the Equation 3.1.3 by A, see Equation 3.1.4, one will observe that the eigenvectors Υ of C are equal to $A\Upsilon_L$. So, it is possible to compute easily the eigenvectors of the covariance matrix C by using a smaller matrix L.

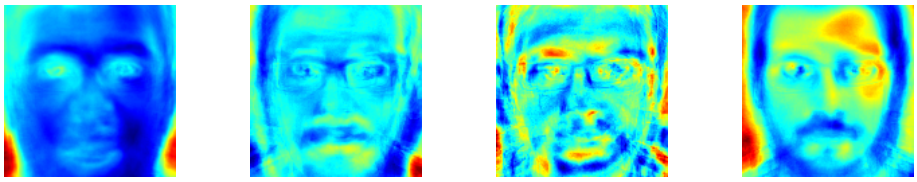
$$AA^TA\Upsilon_L = A\Upsilon_L\Lambda_L \mapsto \underbrace{AA^T}_{C}\underbrace{A\Upsilon_L}_{\Upsilon} = \underbrace{A\Upsilon_L}_{\Upsilon}\Lambda_L \tag{3.1.4}$$

Selection of Eigenvectors

Select M′ eigenvectors $\Upsilon'_L \in \mathbb{R}_{[M \times M']}$ corresponding to the largest eigenvalues of Λ_L as the the principal components, and compute the M′ eigenvectors $\Upsilon' \in \mathbb{R}_{[N^2 \times M']}$ of C by:

$$\Upsilon' = A\Upsilon'_L \tag{3.1.5}$$

These eigenvectors $\Upsilon' \in \mathbb{R}_{[N^2 \times M']}$ have the same dimensionality as the training face images, and they can be seen as face images; in fact, they are called Eigenfaces [TP91] [BHK97]. For example, the (Figure 3.1.2) shows four eigenfaces of a training set of 300 face-images (see some samples in Figure 3.1.1). The M' larges eigenvalues span a M'-dimensional space that best describes the set of training face-images. One can say that these eigenvectors maximize the determinant of the total variance matrix of the projected face images [BHK97].



(a) (b) (c) (d)

Figure 3.1.2: Four Eigenfaces from the sample set of 300 face-images

Projection into the Face-space

Compute the projection of a query face image $\Gamma_{query} \in \mathbb{R}_{[N^2 \times 1]}$ into the M'-dimensional space.

$$\Omega_{query} = \Upsilon'^{T}(\Gamma_{query} - \Psi) \tag{3.1.6}$$

The vector $\Omega_{query} \in \mathbb{R}_{[M' \times 1]}$ describes the contribution of each eigenface in representing the query face image, treating the eigenfaces as a basis for representing face images in the M'-dimensional space; in other words, Eigenfaces method transforms the query image into a descriptor Ω_{query}.

Reconstruction of face-images

Compute the reconstruction $\Phi_{reconstructed} \in \mathbb{R}_{[N^2 \times 1]}$ of the query face image $\Gamma_{query} \in \mathbb{R}_{[N^2 x 1]}$ by projecting the descriptor $\Omega_{query} \in \mathbb{R}_{[M' \times 1]}$ back to the high-dimensional space by:

$$\Phi_{reconstructed} = \Upsilon' \Omega_{query} \tag{3.1.7}$$

For example, The Figure 3.1.3 shows the reconstruction of the face images in Figure 3.1.1.

(a) (b) (c) (d)

Figure 3.1.3: Reconstruction of the face images (see Figure 3.1.1)

Discarding non face-images

Determine whether the query image γ_{query} is a face image or not by comparing the euclidean distance ϵ between the reconstructed face image $\Phi_{reconstructed}$ and the zero-normalized image vector Φ_{query}, see Equation 3.1.8, with a threshold value $\theta_{epsilon}$. If ϵ is smaller than the threshold $\theta_{epsilon}$, the query image will be considered as a face image.

$$\epsilon^2 = \|\Phi_{query} - \Phi_{reconstructed}\|^2 \tag{3.1.8}$$

with $\Phi_{query} = \Gamma_{query} - \Psi$.

Classification of face-images

If the query image is a face image, one will, then, determine which class the query image belongs to. For that end, one computes the Euclidean distance ϵ_k, see Equation 3.1.9, between the descriptor $\Omega_{query} \in \mathbb{R}_{[M' \times 1]}$ and the average descriptors, $\Omega_k \in \mathbb{R}_{[M' \times 1]}$ for $k = 1, 2, ..., K$, of a batch of face images per class from the training dataset with K classes, and one compare it with a threshold θ_k.

$$\epsilon_k^2 = \|\Omega_{query} - \Omega_k\|^2 \tag{3.1.9}$$

If $min(\epsilon_k)$ is smaller than θ_k, then the query face image will belong to class k. Otherwise, if $min(\epsilon_k)$ is greater than θ_k, then the query face image will be classified as unknown; if required, this unknown image could become part of the training set as a new class, in which case, it would be necessary to recompute the eigen- vectors and values [TP91].

As a drawback, the Eigenfaces method does not consider the class labels of the training images, so it includes also intra-class variance—different facial expressions or

viewing directions per class— in the variance matrix C that is unwanted. This intra-class variance is irrelevant information for recognition purposes [BHK97]. Besides, there is not only variation due to differences among faces, but also due to illumination. These variations are even as large as the variations of faces, and they could become eigenvectors of C; thus, they become part of the set Eigenfaces [BHK97].

3.1.3 *FisherFaces*

The Fisherfaces method is another holistic method that tries to overcome the drawbacks of Eigenfaces, see subsection 3.1.2, for creating a face recognition technique that is, particularly, invariant to illumination changes (intensity, and direction of light sources) and face expression variations [BHK97]. As the Eigenfaces method, Fisherfaces method finds a basis of a low-dimensional space for representing face-images, and thus, for executing proper face recognition tasks. Fisherfaces method uses the Linear Discriminative Analysis, which minimizes the intra-class variance, and maximizes the inter-class variance of a set of training face images with a certain number of classes; in general, face images from different classes are separated as far as possible—as Eigenfaces method does—, and face images from the same class are grouped together [BHK97]. Fisherfaces finds an optimal projection by maximizing the ratio between the inter- and intra-class variations of the training face images. For that end, the authors in [BHK97] define two variance matrices: The inter-class variance S_B and the intra-class variance S_W, as follows.

Consider a face image $I(x, y) \in \mathbb{R}_{[N\times N]}$ as a vector $\gamma \in \mathbb{R}_{[N^2\times 1]}$, a training set of M face-images $\Gamma = \gamma_1, \gamma_2, ..., \gamma_M \in \mathbb{R}_{[N^2\times M]}$, and the average face $\Psi = \frac{1}{M}\sum_{m=1}^{M}\gamma_m$. The inter-class variance $S_B \in \mathbb{R}_{[N^2\times N^2]}$

$$S_B = \sum_{k=1}^{K} N_k(\Psi_k - \Psi)(\Psi_k - \Psi)^T \tag{3.1.10}$$

with K the number of classes, N_k number of samples per class k, and $\Psi_k \in \mathbb{R}_{[N^2\times 1]}$ the mean image per class k. And the intra-class variance $S_W \in \mathbb{R}_{[N^2\times N^2]}$

$$S_W = \sum_{k=1}^{K}\sum_{\Gamma_i\in k}(\Gamma_i - \Psi_k)(\Gamma_i - \Psi_k)^T. \tag{3.1.11}$$

The optimal projection Υ_{opt} is chosen as the matrix with orthonormal columns which maximizes the ratio of the determinant of the inter-class variance matrix to the determinant of the intra-class variance matrix of the projected training face-images.

$$\Upsilon_{opt} = \operatorname{argmax} \left| \frac{\Upsilon^T S_B \Upsilon}{\Upsilon^T S_W \Upsilon} \right| \tag{3.1.12}$$

with $\Upsilon_{opt} = [\Upsilon_1, \Upsilon_2, ..., \Upsilon_m] \in \mathbb{R}_{[N^2 \times m]}$ the $m = K - 1$ eigenvectors of S_B and S_W corresponding to the largest $K - 1$ eigenvalues Λ_{opt}, such that:

$$S_B \Upsilon_{opt} = S_W \Upsilon_{opt} \Lambda_{opt} \tag{3.1.13}$$

Unfortunately, the S_W matrix is singular because its rank is at most $M - K$, and the number of M samples is smaller than N^2 ($M << N^2$). So, it is necessary, first, to reduce the dimensionality of the training face-images by projecting them to a lower-dimension space $M - K$ in which S_W is not singular, and second, to apply the Fisher's Linear Discriminant (FLD), see Equation 3.1.12, for reducing them a to $K - 1$-dimension space. This method, in two steps, is formally called Fisherfaces [BHK97]. It follows,

1. Dimensionality reduction: The training face-images are projected to a $M - K$-dimensional space by using the standard PCA, as Eigenfaces does subsection 3.1.2. More compactly written:

   $$\Upsilon_{pca} = \operatorname{argmax} \left| \Upsilon^T C \Upsilon \right| \tag{3.1.14}$$

   with matrix $C \in \mathbb{R}_{[N^2 \times N^2]}$ the covariance of the training set, see Equation 3.1.1, and $\Upsilon_{pca} \in \mathbb{R}_{[N^2 \times (M-K)]}$ the N^2-dimensional eigenvectors of C corresponding to the largest $M - K$ eigenvalues.

2. Standard FLD: Following the Equation 3.1.12

   $$\Upsilon_{fld} = \operatorname{argmax} \left| \frac{\Upsilon^T \Upsilon_{pca}^T S_B \Upsilon_{pca} \Upsilon}{\Upsilon^T \Upsilon_{pca}^T S_W \Upsilon_{pca} \Upsilon} \right| \tag{3.1.15}$$

   with $\Upsilon_{fld} \in \mathbb{R}_{[(M-K) \times (K-1)]}$ the $M - K$-dimensional eigenvectors corresponding to the largest $K - 1$ eigenvalues.

The final projection $\Upsilon_{opt} \in \mathbb{R}_{[N^2 \times (K-1)]}$ is:

$$\Upsilon_{opt}^T = \Upsilon_{fld}^T \Upsilon_{pca}^T \tag{3.1.16}$$

As the Eigenfaces, see subsection 3.1.2, these $K-1$ eigenvectors with the same dimensionality of the training face-images N^2 can be considered, as well as, face-images, called Fisherfaces. The Figure 3.1.4 shows five examples of Fisherfaces from a dataset of 300 face images and 14 classes (persons)— the same examples shown in Eigenfaces, see Figure 3.1.1. These descriptors span a $K-1$ dimensional space, where face-images from the same class (person) are projected closes each other, while face-images from different classes are projected as far as possible [BHK97].

(a) (b) (c) (d) (e)

Figure 3.1.4: Five examples of Fisherfaces

The Fisherfaces method surpasses Eigenfaces. It projects face-images to a low-dimensional space, augmenting the inter-class variance and decreasing the intra-class variation, being easy to compute and robust against lighting and face expressions. However, Fisherfaces descriptors are not robust against partial occlusion.

3.1.4 *Local Binary Pattern*

Holistic methods, for solving the problem of face recognition e.g. Eigenfaces and Fisherfaces, treat input data as a vector in a high-dimensional space, and they extract a lower-dimensional subspace, which preserves discriminative information for recognition [FFL16]. However, this discriminative information might be generated by external sources, or they do not contain spatial information. Besides, methods using the complete high-dimensional image as input require high machine resources, and they are time consuming. As a necessity, extracting local features, from images without using the whole image as a high-dimensional vector, becomes important. Following that point, the authors in [AHP06] present a method for obtaining face descriptors based on local texture features. They use a texture operator in small regions of face-images to obtain, what the authors call, LBP-images, showing that face

images are compositions of textures: flat areas, spots, lines, and edges. Then, they combine the spatial information of the small regions. As a result, they extract face descriptors that are more robust against illumination variation and partial occlusion, and computationally cheaper than EigenFaces and FisherFaces [doc16].

The texture operator, used by [AHP06], is the local Binary Pattern (LBP operator). This operator is a highly discriminative texture descriptor that is invariant to gray level changes, and it is computational efficient [AHP06]. It encodes edge information in a binary vector, by thresholding intensity values of pixels in small squared or circular grids, mostly grids of 3x3 pixels; most specifically, it compares the intensity value of center pixels in grids of an image with intensity values of its pixel neighbors. If the intensity of the center pixel is greater than or equal to its neighbor's, then it will code a '1', otherwise it will code a '0'. These bits are organized in a binary vector with an specific order depending on the direction of comparisons (see Figure 3.1.5). As a result, one gets a binary number for each pixel in the center of the selected grid, e.g 11001111 for a grid 3x3. The binary numbers can be represented as decimal numbers, which are used for creating a new image, called LBP image (see LBP images from the four samples in Eigenfaces Figure 3.1.6).

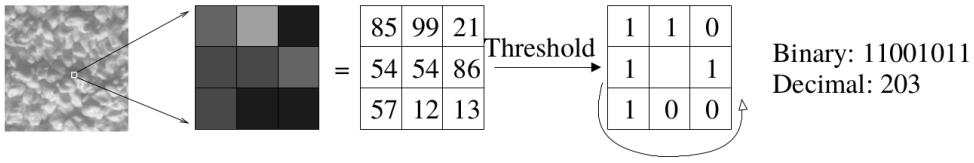
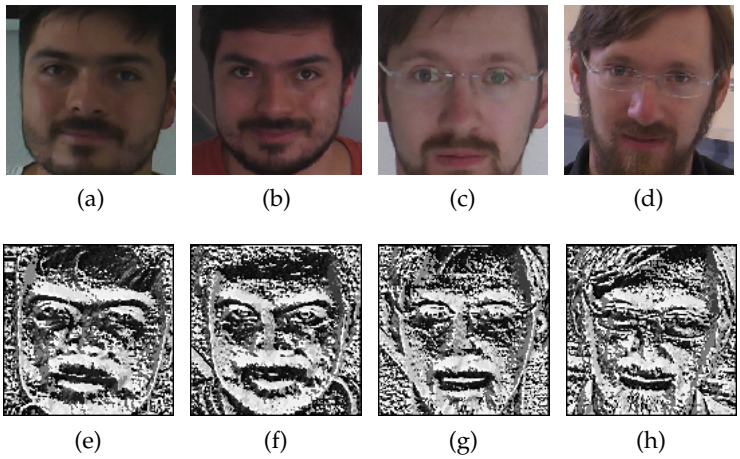


Figure 3.1.5: Example of the LBP operator [AHP06]

To compute the face descriptor, the authors in [AHP06] follow:

1. LBP image computation: One compute a LBP image by applying the LBP operator, see Equation 3.1.17, to center pixels $X_c = (x_c, y_c)$ in a circular grid, given by P number of pixel neighbors $X_p = (x_p, y_p)$ and a radius R, slided through a face-image.

$$LBP(X_c) = \sum_{p=0}^{P} 2^p \cdot S(I_{X_p} - I_{X_c}) \tag{3.1.17}$$



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 3.1.6: Let take some samples from a set of 300 face-images

with,

$$S(x) = \begin{cases} 1 & \text{if} x \geqslant 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.1.18}$$

the image coordinate of the neighbors $X_p = (x_p, y_p)$ for $p = 1, 2, ..., P$ that are computed by,

$$\begin{aligned} x_p &= x_c + R\cos(\tfrac{2\pi p}{P}) \quad p = 1, 2, ..., P \\ y_p &= y_c + R\sin(\tfrac{2\pi p}{P}) \quad p = 1, 2, ..., P \end{aligned} \tag{3.1.19}$$

and I_X the intensity of a pixel X with image coordinates (x, y) [OPM02].

The circular grid's size is determined by the radius R and the number of neighbor pixels P, see Figure 3.1.7. If the computed image coordinates of a neighbor pixel X_p are not integers, one might interpolate its intensity value—e.g. one could use the bilinear interpolation, [doc16].

2. Adding spatial information: The LBP image is divided in m local regions. For each region $i = 1, 2, ..., m$, one computes its histogram H_i, and one builds local

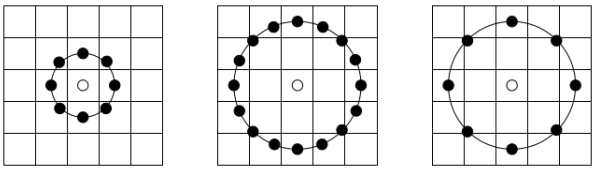


Figure 3.1.7: Different circular grids: $(P = 8, R = 1)$, $(P = 16, R = 2)$, $(P = 8, R = 2)$ respectively [AHP06]

descriptors that are important for keeping the spatial relations in faces as they do not have the same texture. The global or final descriptor is the concatenation of the m histograms $LBPH_I = [H_1, H_2, ..., H_m]$, called Local Binary Pattern Histogram (LBPH) [AHP06]. This final descriptor is more robust against illumination and pose variation than holistic methods, e.g. Eigenfaces or Fisherfaces.

For recognition tasks and having computed the LBPH descriptor for a set of N training face images $LBPH_{training} = [LBPH_1 LBPH_2, ..., LBPH_N]$, one computes distance measurements among the set of training $LBPH_{training}$ and the $LBPH_{query}$ of a query face image, and one selects the persons identity corresponding with the smallest distance [doc16].

3.2 DEEP CNNS FOR VISION TASKS

Having introduced the ANNs (see subsection 2.1.3), CNNs (see subsection 2.1.8), and their learning method (see subsection 2.1.7), this section introduces the deep CNN architectures, which are the basis for the face recognition CNN architecture that is used in this thesis. The next architectures are not thought, specifically, for face recognition, but for general object recognition. Deep CNNs are models that are designed for complex object recognition. These CNN have enormous learning capacities, encoding large and challenging datasets—those datasets consist of millions of images—, showing excellent results in object recognition. They are named deep, because they are composed of a large number of layers.

3.2.1 *AlexNet*

The authors in [AK12] trained a large CNN for ILSVRC−2012 competition (classification of 1.2 million of images in 1000 classes) [RDS+15]. Their CNN has 5 convolutional layers, 3 fully-connected layers, ReLU layers, and 3 overlapping pooling layers, see

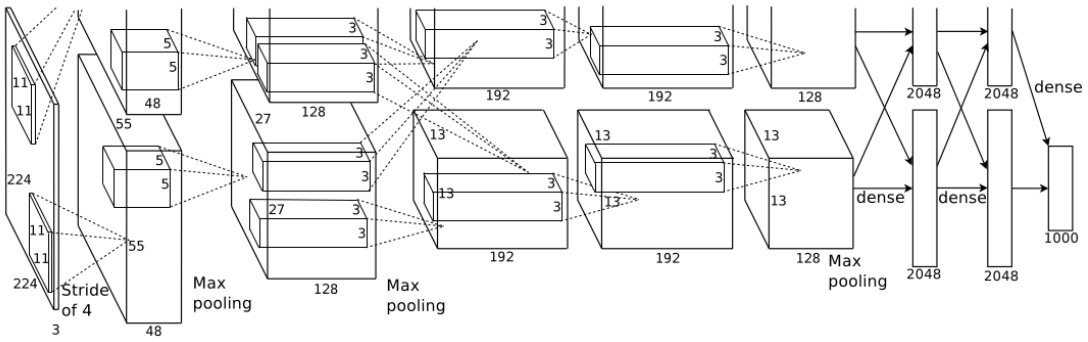


Figure 3.2.1: AlexNet architecture [AK12]

Figure 3.2.1. They were the first to combine CNNs, ReLUs, Pooling, and Dropout in a deep architecture for computer vision, which is, also, trained supervised on a large dataset. They found that a CNN with ReLUs trains six times faster than a CNN with sigmoids or tanh activation functions. Also, the authors utilized a normalisation layer after the first and second convolutional layers, called Local Response Normalisation, that implements a lateral inhibition in layers, helping generalization. Using overlapping pooling layers improved the performance of the Alexnet; moreover, the authors observed that their model does not tend to overfit when using these pooling layers. The last fully-connected layer implements a 1000−way Softmax, see Equation 2.1.17, producing a distribution over 1000 class labels.

The authors divided the filters and their feature-maps in two GPUs. The filters in second, fourth, and fifth convolutional layers are connected just to filters in the same GPU; opposite case, the filters in third convolutional layer and fully-connected layers are connected to their whole input feature-map. The layers and their properties are summarized in Table 3.2.1.

An initial drawback of the Alexnet—and, for now on, a big problem of large CNNs—is the overfitting. This CNN has 60 millions of parameters, and even though one has 1.2 millions of images for training, the number of parameters is huge. So, it is mandatory to add techniques that help CNNs to overcome overfitting. For example, the authors of Alexnet utilized two techniques: Data Augmentation, and Dropout. Data augmentation seems the most obvious choice—the idea is to enlarge the training set. Pointedly, they resized the input training images to $[256 \times 256]$, and they generated $[224 \times 224]$ random translated, and reflected patches from those images. For testing, they resized the input images to $[256 \times 256]$, and they computed 10 predictions: from five $[224 \times 224]$ patches, and their mirroring; specifically, the five patches are taken

Layer	Properties
Input	$[224 \times 224 \times 3]$
1 Convolutional in 2 GPUs	$D = 48$ Filters of size $[11 \times 11 \times 3]$ per GPU, $S = 4$
Local Response Normalization	
Pooling	$S = 2$
2 Convolutional in 2 GPUs	$D = 128$ Filters of size $[5 \times 5 \times 48]$ per GPU
Local Response Normalization	
Pooling	$S = 2$
3 Convolutional in 2 GPUs	$D = 192$ Filters of size $[3 \times 3 \times 256]$ per GPU
4 Convolutional in 2 GPUs	$D = 192$ Filters of size $[3 \times 3 \times 192]$ per GPU
5 Convolutional in 2 GPUs	$D = 128$ Filters of size $[3 \times 3 \times 192]$ per GPU
1 Fully-connected	$D = 2048$ per GPU
2 Fully-connected	$D = 2048$ per GPU
3 Fully-connected (Softmax)	$D = 1000$ per GPU

Table 3.2.1: Alexnet's layers [AK12]

from the four corners and the center of the input image—this testing technique will be studied for face recognition purposes in this thesis. Moreover, they modified the RGB values of the training images; precisely, they computed the eigenvectors and eigenvalues of the training image set by performing the PCA method, and they added multiples of the eigenvectors proportional to the eigenvalues and a random variable. The second technique was the Dropout. It consist in shutting down neurons in the forward pass with a certain probability (mostly 50%). Dropout reduces strongly overfitting; nevertheless, training iterations increase.

Alexnet was trained using SGD, see subsection 2.1.6, with momentum and weight decay, see Equation 2.1.19, with parameters $\gamma_1 = 0.9$, $\gamma_3 = 0.0005$, and a variable γ_1 learning rate starting from $\gamma_2 = 0.01$ and decreased by 10, when the validation error stopped decreasing. Alexnet achieved top-1 test error rate of 37.7%, and top-5 test error rate of 17.0%, becoming the best model of the ILSVRC-2012 competition [RDS+15] with almost 10% difference with the second and third models of the competition.

3.2.2 *VGG*

The authors in [SZ15] studied the performance of CNNs when increasing their depth; that means, when CNNs have many convolutional layers. They trained six CNNs

VGG configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
Input layer $[224 \times 224 \times 3]images$					
1 convs (D=64, F=3)	1 convs LRN (D=64, F=3)	2 convs (D=64, F=3)	2 convs (D=64, F=3)	2 convs (D=64, F=3)	2 convs (D=64, F=3)
Max Pooling					
1 convs (D=128, F=3)	1 convs LRN (D=128, F=3)	2 convs (D=128, F=3)	2 convs (D=128, F=3)	2 convs (D=128, F=3)	2 convs (D=128, F=3)
Max Pooling					
2 convs (D=256, F=3)	2 convs LRN (D=256, F = 3)	2 convs (D=256, F=3)	2 convs (D=256, F=3) 1 conv (D=256, F=1)	3 convs (D=256, F=3)	4 convs (D=256, F=3)
Max Pooling					
2 convs (D=512, F=3)	2 convs LRN (D=512, F = 3)	2 convs (D=512, F=3)	2 convs (D=512, F=3) 1 conv (D=512, F=1)	3 convs (D=512, F=3)	4 convs (D=512, F=3)
Max Pooling					
2 convs (D=512, F=3)	2 convs LRN (D=512, F = 3)	2 convs (D=512, F=3)	2 convs (D=512, F=3) 1 conv (D=512, F=1)	3 convs (D=512, F=3)	4 convs (D=512, F=3)
Max Pooling					
Fully connected (D=4096)					
Fully connected (D=4096)					
Fully connected (D=1000)					
Softmax					

Table 3.2.2: VGG's layers [SZ15]

with different depths (two CNNs with 11, one with 13, two with 16, and one with 19 convolutional layers); in which case, it was necessary to reduce the size of the neuron's receptive field F ($F = 3$ for convolutional layers and $F = 2$ for pooling layers) and the stride S ($S = 1$ for convolutional layers and $S = 2$ for pooling layers) in comparison with Alexnet; besides, they utilized zero-padding for preserving the spatial resolution. The six CNNs are shown in Table 3.2.2, with ReLUs following each convolutional layer, and LRN for Local Response Normalization (as in Alexnet).

The authors proposed adding more adjacent convolutional layers with small receptive fields, since the effective receptive field of those layers is the same as the single convolutional layer's with large receptive fields. However, one adds, with those layers, also non-linearities (ReLUs), which, as the authors say, helps for better discrimination; moreover, convolutional layers with small receptive fields have lesser number of parameters than the ones with large receptive fields. One can see this technique as a

decomposition of a convolutional layer to few convolutional layers but with smaller receptive fields.

The VGG was trained following the same configuration of the Alexnet: Batch GD with momentum and weight decay, see Equation 2.1.19, with parameters: momentum $\gamma_1 = 0.9$, weight decay $\gamma_2 = 0.0005$ and three learning rates $\gamma_3 = [10^{-2}, 10^{-3}, 10^{-4}]$. They trained, first, the shallowest CNN (A) with random weights initialization; that is, the initial weights of the convolutional layers are random values, which are drawn from a normal distribution with zero mean and 10^{-2} variance. Then, they trained the other five CNNs with the A-CNN's weights as initial values. During training, a CNN was fed with random crops of size [224x224x3] that were taken from rescaled images of the training set; they were randomly horizontal flipped, and randomly RGB colour shifted, following the same data augmentation technique that [AK12] introduced. The authors considered two rescaling approaches: rescaling images, such that their smallest side is a fixed value—they used two sizes: 256, and 384—, or their smallest side is randomly drawn from a range $[256, 512]$.

For testing, the authors followed two techniques. In the first one, rescaled test-images and their horizontal flips were fed to a CNN. The fully-connected layers of this CNN, however, were transformed to convolutional layers; concretely, the first, second, and third fully-connected layers were transformed to convolutional ones with receptive fields of sizes $F = 7$, 1 and 1 respectively. For computing the final score vector, one utilized a sum-pooling on the final feature map (third fully-transformed layer's output), and one averaged the final score vectors of the rescaled test-images and their horizontal flips. The second technique consists in averaging the final score vectors from multiple crops of test-images, introduced by Alexnet, but also, the authors of VGG included crops over three scales. 25 crops of size $[224x224 * 3]$ over three scales (150 crops in total) were used for computing a single test-image's final score vector. The averaging of CNN's descriptors from multiple test-image crops will be studied for face recognition purposes in this thesis.

In the ILSVRC 2014 competition [RDS+15], the authors of VGG achieved top−1 and top−5 test error rates of 25.5% and 8.0%, respectively, with the deepest CNN (E), which is 12.2% for top−1 and 9% for top−5 better than the Alexnet's test error rates; even, the shallowest CNN (A) reached 39.6% top−1 and 10.4% top−5 test error rates, being also superior to the Alexnet's. Therefore, the authors concluded that the performance of CNNs increases when they are deep architectures with small receptive fields for convolutional layers.

4

METHOD

Previously, CNNs and deep CNNs architectures for general object recognition tasks have been introduced. These architectures encode large datasets, and show excellent results in challenging object recognition tasks. One of those tasks is, specifically, the face-recognition. As large number of face-image datasets are today available, CNNs and deep CNNs are suitable to learn that amount of information, such that they can, accurately, classify a set of face-images, or obtain a low-dimensional representation of face-images [TYRW14]. This chapter covers the use of deep CNNs architectures for face recognition and, in addition, face-verification tasks. Firstly, a deep CNN architecture, used concretely for face-recognition and -verification purposes, is introduced. Secondly, the face-recognition and -verification tasks using this specific deep CNN, focusing, mainly, on the testing technique, used similarity measurements, and the extreme occlusion performance, are discussed. And Thirdly, the approaches, for improving the performance of the aforementioned CNN, are introduced. This approaches are the focus of this thesis.

4.1 DEEP FACE RECOGNITION

Deep Face Recognition (DFR) [PVZ15] is inspired by the Alexnet's (see subsection 3.2.1), and the VGGNet's (see subsection 3.2.2) success in object recognition. The authors evaluated three CNN architectures and two classifiers on face recognition and verification tasks. They selected, specifically, the A, B, and D CNNs of VGG, see Table 3.2.2, and utilized two objective functions for training two different classifiers: an N-way classifier, and an, what the authors called, L-dimensional metric embedding classifier. The Figure 4.1.1 shows the general DFR architecture.

4.1.1 *N-way classification*

The first objective of DFR was to recognize face-images of different individuals, when a training face-image set with N identities was given; concretely, a training set of $N = 2622$ identities was used, the VGG dataset see subsection 5.1.1. The authors utilized the Softmax-log loss, see Equation 2.1.17, as the objective function for training

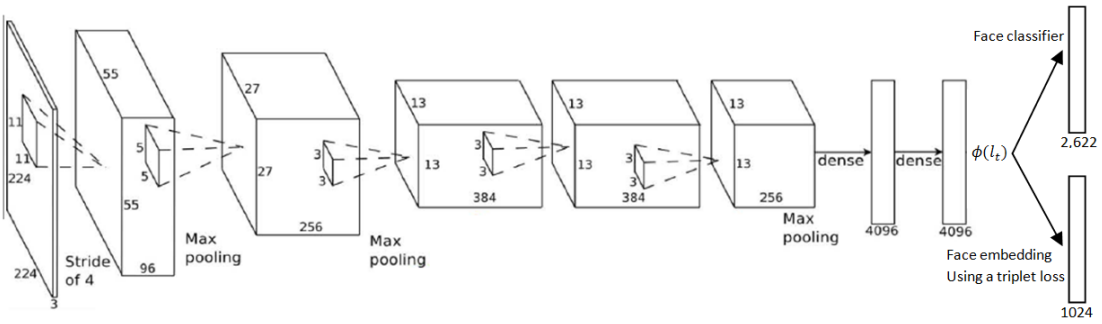


Figure 4.1.1: DFR general architecture [PVZ15]

the DFR architectures, where the final score vectors are compared with the ground-truth class identity vectors; such that, the last fully-connected layer implements an "N" linear classifier. This layer learns a projection $W \in \mathbb{R}_{[N\times D]}$ that computes a score vector $x_t \in \mathbb{R}_{[N\times 1]}$ for an image I_t.

$$x_t = W\phi(I_t) + b \in \mathbb{R}_{[N\times 1]} \tag{4.1.1}$$

with $\phi(I_t)$ being the D−dimensional output vector from the second fully-connected layer, see Figure 4.1.1.

4.1.2 *Face Verification*

The second objective was to apply the learned knowledge about faces for verification purpose, answering the question whether two face-images show the same person. Therefore, the D-dimensional vector $\phi(I_t)$ is used as a descriptor or as a low-dimensional representation of face-images for face identity verification. For this task, one compares two face-images $[I_1, I_2]$ by means of the Euclidean distance between their descriptors $[\phi(I_1), \phi(I_2)]$. If the distance is smaller than a threshold θ, then two face-images will be considered as belonging to the same person.

4.1.3 *L-dimensional metric embedding classifier*

The D-dimensional descriptors $\phi(I_t)$ are still comparably high-dimensional vectors. So, the authors of DFR improved them by projecting them to a lower-dimensional space L ($L < D$), such that, ideally, descriptors from the same face-image identity

are grouped nearer than descriptors from different face-image identities [PVZ15]. Purposely, the last fully-connected layer is exchanged with another one that is trained by minimizing a different objective function, see Figure 4.1.1. This objective function is, precisely, the empirical triplet loss [FS15], see subsection 4.1.4, which finds the optimal weights $W' \in \mathbb{R}_{[L\times D]}$. The weights W' project the $\phi(I_t)$ to a low-dimensional descriptor $x_t \in \mathbb{R}^L$ as follows.

$$x_t = W' \frac{\phi(I_t)}{\|\phi(I_t)\|_2} \in \mathbb{R}_{[L\times 1]} \tag{4.1.2}$$

The goal was to derive final descriptors x_t that are more distinctive, more compact, and better for face-verification tasks than the descriptors $\phi(I_t)$ [PVZ15].

4.1.4 *Empirical Triplet loss*

One wants to ensure that descriptors of face-images from the same person are closer than any descriptor of face-images from another person. For that purpose, the authors in [FS15] proposed the empirical triplet loss as an objective function for the Gradient Descent, see subsection 2.1.6. By using this objective function, one minimizes the distance among descriptors of the same persons (positive descriptors), and, at the same time, one maximizes the distance among descriptors from different persons (negative descriptors); such that, distances to negative descriptors should be larger than a certain margin α, see Figure 4.1.2 [FS15].

For a training triplet set T with anchor-descriptors a, positive descriptors p $(p \neq a)$, and negative descriptors n, the empirical triplet loss, to be minimized, is:

$$E(W') = \sum_{(a,p,n)\in T} max\left\{0, \alpha - \|X_a - X_n\|_2^2 + \|X_a - X_p\|_2^2\right\} \tag{4.1.3}$$

with α a positive triplet margin, and X_t a L_2-normalized and projected (using the trained W') vector of $\phi(I_t)$:

$$X_t = W' \frac{\phi(I_t)}{\|\phi(I_t)\|_2} \quad W' \in \mathbb{R}_{[L\times D]} \tag{4.1.4}$$

4.1.5 *Training*

The authors in [PVZ15] followed the same training scheme of Alexnet and VGG: Batch GD with momentum and weight decay, see Equation 2.1.19. The following

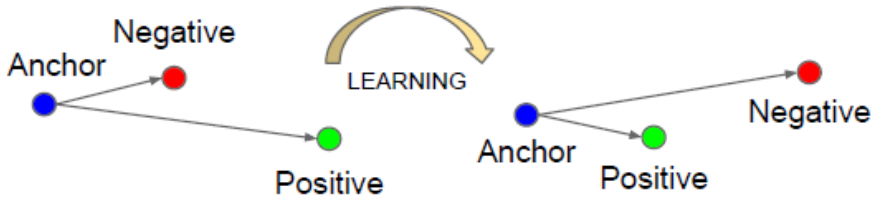


Figure 4.1.2: Triplet loss learning goal [FS15]

parameters were used: momentum $\gamma_1 = 0.9$, weight decay $\gamma_2 = 0.0005$ and three learning rates $\gamma_3 = [10^{-2}, 10^{-3}, 10^{-4}]$. They trained, first, the shallowest CNN (A) with random weights initialization from a normal distribution $(\mu = 0, \sigma = 10^{-2})$. Then, they fine-tuned B-, and D-CNNs using the parameters of the shallow A-CNN as the starting point. During training, random-cropped, and -horizontal flipped face-images of size [224x224x3] over resized face-images of 256 px from the VGG training set, see subsection 5.1.1, are fed to the CNNs; contrary to Alexnet and VGG, the face-images are not colour shifted. For learning the L-dimensional metric embedding (see subsection 4.1.3), a CNN is frozen and its last fully-connected layer is replaced, then, this layer is learnt by using stochastic GD feeding triplets (a, p, n) from the training set of the target domain— this training set refers to all the face-images from LFW dataset that do not belong to the unrestricted setting of the dataset subsection 5.1.2— that violate the triplet margin α; that is, distances among anchor a and positive p descriptors are superior to α, and distances to negative n are inferior to α.

4.1.6 *Testing*

For the face verification task, the authors in [PVZ15] followed the second testing technique of VGG network—averaging the final score vectors from multiple crops of test-images—, see subsection 3.2.2, and, also, the standard evaluation protocols ("unrestricted setting") of LFW dataset, see subsection 5.1.2, and YTF dataset, see subsection 5.1.3. For testing on the LFW dataset, they evaluated 3000 pairs of face-images— a pair consists of two face-images of the same person—, and 3000 non-pairs face-images. For each image of this testing set, they took the average of 30 descriptors as the final representation, or as a global face-descriptor of a single face-image. The 30 descriptors are, specifically, the D-dimensional vector $\phi(I_t)$ when feeding $[224 \times 224 \times 3]$ crops extracted from the four corners and the center of $(256, 384, 512)$ resized face-images and their horizontal flips to the CNN. In the same way, for testing on the YTF dataset, they evaluated, in this case, 2500 pairs of face-videos, and 2500

non-pairs face-videos. For each face-video, they ordered the frames by their facial landmark confidence score, and they selected the top K frames, whose K descriptors $\phi(I_t)$ are computed and averaged. The average of those K descriptors represents a single face-video. The authors compared two face-descriptors by using the Euclidean distance. If the distance between the final descriptors from two face-images is smaller than a threshold, the two face-images will belong to the same identity. In addition, the authors in [PVZ15] detected faces on the images of the LFW dataset by using the Deformable Parts Model (DPM), see [MBPVG14]. Since, the face-images of LFW dataset, see Figure 5.1.2, have a bigger bounding-box than the face-images of VGG dataset, see Figure 5.1.1.

4.2 DEEP FACE RECOGNITION DISCUSSION

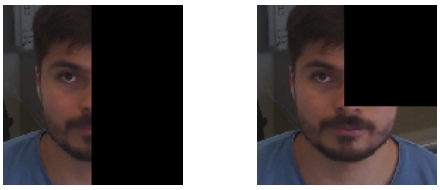
The multicropping over three sizes—originally thought for avoiding overfitting and for multisize robustness since objects can be of different sizes— is beneficial for general object recognition. Nevertheless, this testing technique implemented by [PVZ15], following the guidelines of the first deep CNN (Alexnet), is not optimal for face recognition and verification tasks. Some of the 30 $[224 \times 224 \times 3]$ crops, and their horizontal flips of single face-images might not be, exactly, considered as faces, and they might vary the final descriptors, see Figure 4.2.1.

(a) (b)

Figure 4.2.1: Crops of a 256, 384, and 512 resized face-image

Furthermore, extreme occlusion in face-images was not completely studied in DFR. The training and test datasets, VGG, LFW and YTF datasets, are collections of images extracted from internet, and they might have face-images with partial occlusion, due to, mainly, glasses, beards, and, hairstyle changes see Figure 4.2.1. However, human perception needs less visual information coming from objects to recognize them.

Humans can recognize faces under extreme occlusion conditions, see Figure 4.2.2; for example, we can recognize persons just by looking to small portions of their faces.



(a) (b)

Figure 4.2.2: Example of extreme occlusion

Finally, one notices that the previous CNNs (Alexnet, VGG and DFR) utilized only the L_2 norm or Euclidean distance as similarity measurement. Nevertheless, the performance of the L_2 norm deteriorates for comparisons in high-dimension spaces [CC01].

4.3 EXTENTIONS TO DEEP FACE RECOGNITION NETWORK

Based on the previous discussion, different approaches for testing/training, overcoming the noticed disadvantages of DFR, are introduced next. These approaches address to study the performance of CNN when utilizing different similarity measurements of descriptors, to overcome the necessity of using multicropping, and to improve the performance of CNNs under extreme occlusion conditions.

4.3.1 *Spatial Pyramid Pooling layer*

The authors in [HZRS14] pointed out that convolutional layers do not need fixed-size feature-map inputs, since they perform a convolution operation and their filters are not fully-connected to their inputs. However, the fully-connected layers need, necessarily, fixed-size feature-map inputs. In fact, the last convolutional layer is the only one that should generate a fixed-size feature map because the first fully-connected layer is connected to it. For that reason, the authors in [HZRS14] replaced the last max-pooling layer, see Figure 2.1.8, with a new layer to eliminate the need of fixed-size input images— since Alexnet, one uses $[224 \times 224 \times 3]$ cropped images. This new layer is called the spatial pyramid pooling (SPP), whose idea comes from the spatial pyramid matching used originally as an extension of Bag of Features [LSP06].

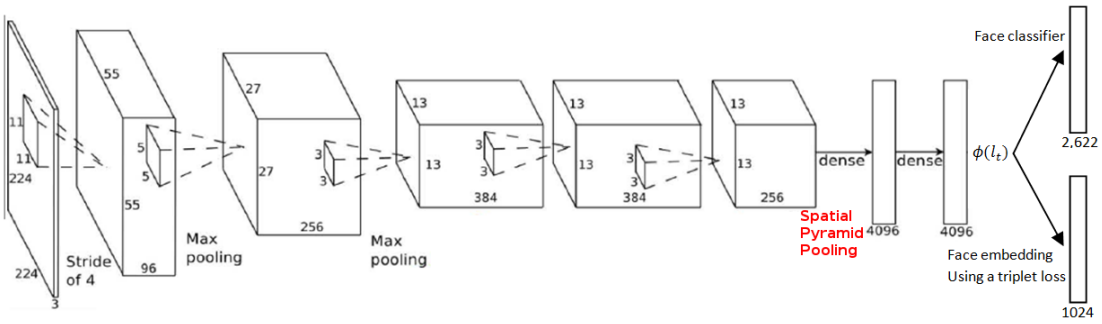


Figure 4.3.1: D-CNN with SPP layer

The SPP layer divides an input from finer to coarser levels; it aggregates their information generating local outputs; and it concatenates these outputs into an overall feature output, see Figure 4.3.2. Its advantage lies on the generation of fixed-size outputs from multisized inputs, and, the most important characteristic for this thesis, it maintains spatial information. More concretely in CNNs, a l-level SPP layer divides a feature-map input, whose depth is D, into $D \cdot M$ spatial bins with $M = \sum_1^l 2^l$, where the level l denotes the number of divisions along the spatial dimensions of the feature-map [LSP06]. Then, the SPP layers concatenates the maximal or the average value of each bin into a vector, which has a fixed-size. This vector becomes the feature-map of the first fully-connected layer, see Figure 4.3.2. By using a SPP layer, the CNN's input can be of any scale.

As the training and testing techniques of DFR, see section 4.1, used multicropping over three scale resized images, the performance on multiscaled face-images of a CNN, when using an SPP layer for avoiding multicropping, is studied. Besides, the performance of the CNN with an SPP layer, when inputs are images with extreme occluded faces, is observed. Precisely, it is sought to study whether CNNs with an SPP layer can discriminate between two faces when certain percentage of the face is occluded in the images. Besides, as the faces are symmetric, using half of a face-image, for obtaining a lower-dimensional global representation, is studied. For an l-level SPP layer, one can mirror the activated bins of its output, obtaining a symmetrical feature-map that is, then, fed to the first fully-connected layer.

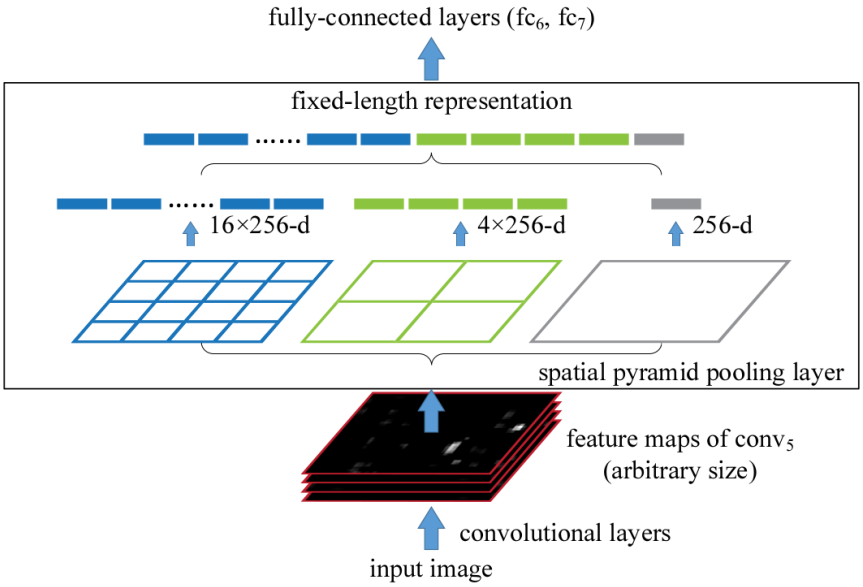


Figure 4.3.2: Spatial Pyramid Pooling SPP layer in a CNN [HZRS14]

4.3.2 *Empirical Triplet loss for partially occluded face-images*

As discussed in section 4.2, humans can recognize faces under extreme occlusion conditions; for example, humans can identify, easily, a person just by looking at one half of his/hers face. Following the same idea of the L-dimensional metric embedding classifier of DFR, see subsection 4.1.3, one wants to ensure that descriptors of half-face images are closer to descriptors of non-occluded face-images of the same person than any descriptor of face-images from other persons. By training the last fully-connected layer minimizing the Empirical Triplet Loss with descriptors of half-face images as anchors, see subsection 4.1.4, one will expect to have a projection W' that will compute a compact- and robust-, against occlusion, final descriptor $x_t \in \mathbb{R}_[N \times 1]$, see Equation 4.1.2.

4.3.3 *Measurements*

In high-dimensionality, the concept of distance, and nearer neighbor may not be qualitatively meaningful [CC01]. Traditionally, one uses the L_2 norm, also called Euclidean distance, as default for spatial (2D or 3D) applications. However, its performance

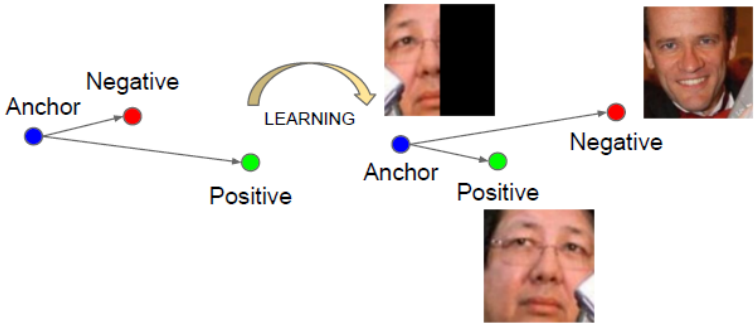


Figure 4.3.3: Triplet Loss with halfoccluded faceimages as query images [HZRS14]

deteriorates rapidly for high-dimensions. For that reason, the performance of four similarity measurements for face-verification with CNNs is studied in this thesis. Since, the CNN's final descriptors are of dimension 4096, and the Alexnet, VGG and DFR networks utilized only the Euclidean distance for testing purposes. The four similarity measurements are: the L_2, the L_1, the cosine, and the Bray-Curtis distances.

L_k *norm*

The commonly used L_k norms are defined as,

$$L_k(x,y) = \sum_{i=1}^{d} (\|x_i - y_i\|^k)^{1/k} \quad with \quad x,y \in \mathbb{R}^d \quad and \quad k \in \mathbb{Z}, \tag{4.3.1}$$

The L_2 norm or Euclidean distance, and the L_1 or Manhattan distance are two examples of L_k norms. The authors in [CC01] showed that L_k norms with low values of k perform better for comparisons among vectors of high-dimensionality.

Cosine distance

The cosine distance measures the cosine of the angle between two non-zero vectors. If the measured cosine is "1", then the two vectors are oriented towards the same direction; that is, the angle between them is $0°$. If the cosine is "0", then the vectors are perpendiculars. On the other side, if the measured cosine is "-1", then the two vectors are opposite. For two vectors (x,y), the cosine distance is defined as:

$$\cos(\theta) = \frac{x \cdot y}{\|x\| \, \|y\|} \tag{4.3.2}$$

Bray-Curtis Dissimilarity

The Bray-Curtis dissimilarity (BC) is a dissimilarity measurement or a non-euclidean distance— it is not a true distance, since it violates the triangle inequality—, normally, used in ecology and biology [BC57], which gives a value to the dissimilarity between two vectors. It outputs an index between zero and one: zero for similar vectors, and one for dissimilar ones. For two vectors (x, y), the BC dissimilarity is defined as:

$$BC(x, y) = \frac{\sum_i |x_i - y_i|}{\sum_i x_i + y_i} \tag{4.3.3}$$

The BC dissimilarity works well for histogram representations [SF15] and high-dimensional vectors, having better performance than L_2, L_1, cosine and the χ^2 distances [Blo81].

4.4 CONCLUSION

Briefly, the idea is to replicate the DFR's results, and to evaluate the influence of the aforementioned approaches on performing face-verification on two testing datasets. Concretely, two deep CNNs are used: the A architecture of DFR, see section 4.1, and a similar CNN with a SPP layer, see subsection 4.3.1, instead of the last pooling layer. Several multicropping configurations on non-, extreme- and partially-occluded face-images are utilized. Moreover, for boosting the performance on extreme-occlusion, an additional fully-connected layer is trained using the empirical triplet loss, in which full face- and partially-occluded face-images of the same identities are grouped together. As an extension, the previously introduced four distances measurements as similarity metrics for the task are used.

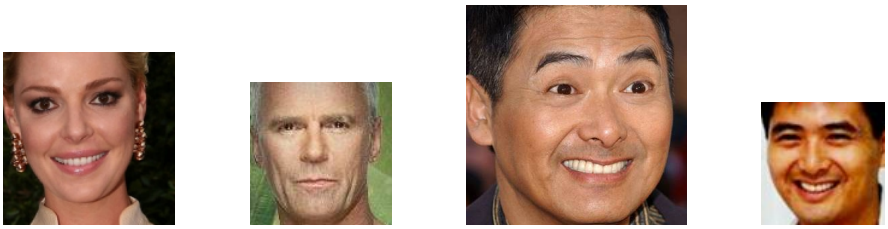
5 EXPERIMENTS

The main task in this thesis is to perform face verification following the standard evaluation protocols ("restricted configuration") of the LFW and YTF datasets. For that purpose, descriptors $\phi(I_t)$ of face-images, which are organized in pair and non-pair groups, are computed using the deep CNN architecture (the A architecture in Table 3.2.2, of subsection 3.2.2), as well as, the proposed one including the SPP layer, see subsection 4.3.1, as descriptor extractors. The similarities between pair-descriptors, and between non-pair descriptors are calculated using different measurements. In the end, these similarity values are compared with a threshold for determining if they belong to the same identity or not. Depending on the dataset, the number of pairs and non-pairs, and the approach for computing a single face-descriptor change. For the LFW dataset, the restricted setting has 3000 pairs and 3000 non-pair face-images. A single descriptor is the average of a a group of descriptors, which are computed from different configurations of crops—mainly, from $[224 \times 224 \times 3]$ crops over different scales of a face-image. For the YTF dataset, its restricted setting has 2500 pair and 2500 non-pair face-videos. In this case, a single face-descriptor is the average of K face-frames of the video. Besides, the performance of the aforementioned deep CNNs, and the previously discussed approaches, see section 4.3, for verification tasks with face-images, or -videos with extreme- and proportional occlusion are evaluated.

This chapter covers, firstly, an introduction of the three datasets and the evaluation metric that was used in DFR [PVZ15] and in this thesis, secondly, the experiments and results of the discussed CNN, and CNN with an SPP layer for face-verification on the restricted settings of the datasets using different multicropping configurations with non-, extreme- and proportional-occlusion are presented.

5.1 DATASETS

Three datasets are used by the DFR network [PVZ15] and this thesis. The VGG dataset is used for training the deep CNN networks for face-recognition, and two benchmark datasets (the LFW and the YTF datasets) are used as testing datasets, allowing direct comparisons with previous face-verification methods.



(a) (b) (c) (d)

Figure 5.1.1: Some samples of VGG dataset [PVZ15]

5.1.1 *Visual Geometry Group Face Dataset*

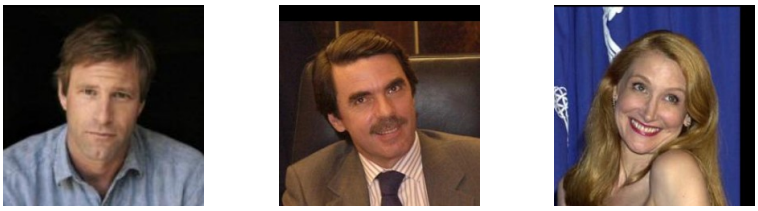
The Visual Geometry Group Face Dataset (VGG dataset) is a large collection of face-images extracted from the Internet. It contains 2.6 million of face-images from 2622 identities; specifically, celebrities and politicians, see Figure 5.1.1. This dataset was obtained following the procedure proposed by [PVZ15], which idea was to assemble a large dataset of labeled face-images with small label noise, and minimal manual annotation; moreover, this dataset does not contain overlapping identities with standard benchmark datasets (LFW, YFT), so it is suitable for training.

Unfortunately, the dataset is available in a text file, which contains URLs to images and their corresponding bounding-box of the detected faces. Due to the constant Web page changes, it was not possible to download all the 2.6 millions face-images. In the end, around 2 million of the images can be obtained, being, still, a large amount of face-images.

5.1.2 *Labeled Faces in the Wild dataset*

The Labeled Faces in the Wild (LFW) is a standard benchmark dataset for face verification. It contains 13233 face-images from 5749 identities extracted from the Internet. Faces in images were detected using the Viola-Jones face detector [GHLM07]. Faces are roughly centered, contain lesser noise but larger bounding-box than the VGG dataset. Besides, faces have different poses, face-expressions, gender, ethnicity, and hairstyles, becoming a challenging benchmark for face-identification purposes.

The authors in [GHLM07] provide two training and testing configurations:



(a) (b) (c)

Figure 5.1.2: Some samples of LFW dataset [GHLM07]

Restricted Configuration

The training and testing information of matched and non-matched face-images is provided in a pairs ave. The file is divided in 10 sets (9 for training and the last one for testing purposes). The pairs ave has in total 3000 matched face-image pairs; that is, pairs of face-images that belong to the same person. And, also, it has 3000 non-matched face-image pairs. This configuration is called restricted, because face-image identities should not be used for training purposes.

Unrestricted Configuration

In addition to the restricted configuration, the authors in [GHLM07] provide a list with the number of face-images per identity. In that case, one can do as many pair and non-pairs as wanted. This list is also divided in 10 sets, from which 9 are used for training and the rest for testing.

As the training dataset used by [PVZ15] is the VGG dataset, both configurations of LFW dataset can be used as the testing dataset. For training the Embedding with triplet loss subsection 4.1.4, authors in [PVZ15] followed the unrestricted configuration.

5.1.3 *YouTube Faces Dataset*

The YouTube Faces (YTF) is a standard benchmark for face verification in video. It contains 3425 videos from 1595 identities, with an average of 2.15 videos per identity, and 181.3 frames per video [WHM11]. The authors detected faces in videos using the Viola-Jones face-detector. For this thesis and following the same approach of [PVZ15], faces were detected in all the video-frames using the DPM detector [MBPVG14], since bounding-boxes that are detected by the Viola-Jones detector are larger than the

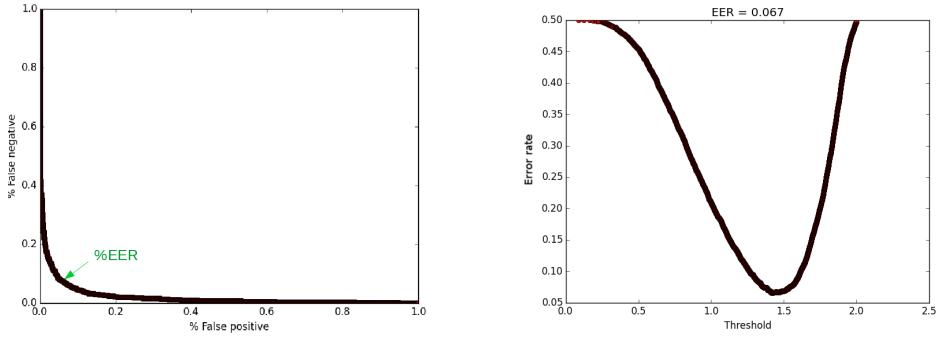
Figure 5.1.3: YouTube Faces [WHM11]

training VGG dataset ones. In the end, each video has as minimum of 45 face-frames, which are used for representing a single face-video.

Same as the LFW dataset, the authors of YTF dataset [WHM11] provide two training configurations: a restricted and an unrestricted configurations. For the restricted configuration, the authors provide a list with 2500 pair face-videos and 2500 non-pair face-videos, which is divided in 10 mutually exclusive sets. This configuration constraints the information for training purposes; that is, only information of same or not same should be used. The unrestricted configuration allows to use video's identities for training purposes, and to create as many pairs and non-pairs as wanted.

5.2 EVALUATION METRIC

As explained before, the face-verification's goal is to tell whether two face-images belong to the same identity. For that, the distance between their descriptors $\phi(I_t)$ is compared with a threshold θ. If the distance is smaller, the face-images will be classified as the same identity. However, the threshold value is not, clearly specified, and it is not learned, directly, by the CNNs. Moreover, the similarity measurements in subsection 4.3.3 provide only information of the distance of two descriptors, but they do not tell, concretely, if two vectors are far or near. Ideally, descriptors of face-images from the same person should be similar, and their distance should be always smaller than distances to descriptors of any other person; that is, there exists a threshold θ that separates the area where descriptors of the same person lie in the face-space. This threshold value should be valid for all the identities in the testing dataset. However, in real applications, there exist cases where descriptors of different persons are nearer than the same person ones [GEAR13]—the area corresponding to descriptors of a person is crossed by other descriptors—, so the verification method accepts a descriptor into a wrong identity, or as called in the literature, a false acceptance. In that case, the threshold θ is too large. Contrary, if the threshold is too small, there will not exist false acceptance, but correct descriptors are rejected, or a false rejection. Therefore, in general, the verification methods show an error. This error is, indeed,



(a) (b)

Figure 5.1.4: False negative vs False positive curve, and the error rate vs Threshold curve [GEAR13]

a metric that tells how accurate methods are. The International Organization for Standardization ISO/IEC 19795-1 proposed several metrics for evaluating a verification method [GEAR13]. One of those metrics, used for face-verification purposes [PVZ15] [KBBN09] [TYRW14], is the Equal Error Rate %EER. The %EER is the value where the False Acceptance Rate (FAR), and the False Rejection Rate (FRR) are equivalent; that means, one selects a threshold from a range of possible values that minimizes the number of false acceptance and, at the same time, the number of false rejection. This can be seen, concretely, in the curve FAR vs FRR (Figure 5.1.4), called ROC curve [GEAR13], for all the possible thresholds.

For an optimal threshold θ_{opt},

$$\theta_{opt} = \mathrm{argmin}(\|FAR(\theta) - FRR(\theta)\|), \forall\theta \in [\theta_{min}, \theta_{max}] \tag{5.2.1}$$

,the %EER is defined as

$$\%EER = \frac{FAR(\theta_{opt}) + FRR(\theta_{opt})}{2} \tag{5.2.2}$$

where the FAR, and the FRR, for a threshold θ, are,

$$FAR(\theta) = \begin{cases} \frac{\sum_{d \in N} 1}{\sum_{c=1}^{n} c} & d \leqslant \theta \\ \frac{\sum_{d \in P} 1}{\sum_{c=1}^{p} c} & d > \theta \end{cases} \tag{5.2.3}$$

with N the set of distances between non-pair descriptors, and P the set of distances between pair descriptors. The advantage of this metric lies in the comparison among different verification methods, since it does not depend on the a specific threshold θ [PVZ15].

5.3 EXPERIMENTS AND RESULTS

The experiments and results for performing face-verification task using CNNs on the testing datasets (LFW, and YTF datasets) with non-, extreme, and proportional occlusion are described and discussed next. For all the experiments, the four similarity measurements, see subsection 4.3.3, are used, and their performances for comparing between descriptors are presented. For both testing datasets, the restricted configuration is chosen.

5.3.1 *Replication of the original publication*

The testing technique of DFR, see section 4.1, for both restricted configurations of the LFW, and YTF datasets, is replicated. Accordingly, the descriptors $\phi(I_t)$ for each of the pairs and non-pairs are computed, and they are compared using the Euclidean distance. For the LFW dataset, a face-descriptor is the average of 30 descriptors $\phi(I_t)$ per face-image, where the CNN computes each of those descriptors from $[224 \times 224 \times 3]$ crops of the four corners and the center of $[256, 384, 512]$ resized face-images, and their horizontal flips. For the YTF dataset, a face-descriptor is the average of K descriptors $\phi(I_t)$, which are computed by the CNN from K face-frames per face-video; specifically, one took $K = 45$, since there are as minimum 45 face-frames in all the face-videos of the YTF dataset. The faces in images or frames of both datasets were detected using the DPM detector [MBPVG14], as [PVZ15] suggested, since the bounding-boxes of the images of the LFW—faces are detected using the Viola-Jones detector [GHLM07]—, and YTF datasets are larger than the ones from the images of the VGG dataset. Furthermore, using the DPM detector, one filters the face-frames in videos, such that the K face-frames, for YTF dataset testing, corresponds to centered faces. This is different from the original publication [PVZ15], because the authors used the face-frames that have the best facial landmark scores.

Figure 5.3.1 shows the relation between %EER and all the four similarity measurements: the L_1, L_2, cosine and Bray-Curtis distances for the restricted configuration of LFW dataset, including, also, the VGG original publications's %EER for the A-CNN architecture (red line) [PVZ15]. Notice the difference of 1.2% between the original

publication's %EER for L_2, and replicated one (L_2's %EER), which are based on the same testing configuration. Besides and among the replication results, the BC distance, clearly, shows better performance for the task, showing a reduction of the %EER of 1.07% with respect to its counterpart (L_2).

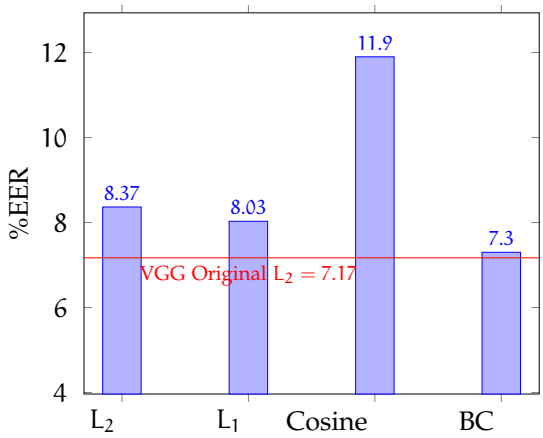


Figure 5.3.1: %EER vs distance measurements for the restricted configuration of LFW dataset

Figure 5.3.2 shows the relation between %EER and all the similarity measurements for the restricted configuration of YTF dataset, including, the DFR original publication's %EER (red line); particularly, the authors in [PVZ15] used $K = 100$ face-frames that were ordered by their facial landmark confidence score [PVZ15]. Following the results of LFW dataset, using the BC distance improves the performance of the testing in comparison to L_2 distance; exactly, the BC shows an improvement of 2.32%. However, the BC's %EER is 1.56% inferior with respect to the original publication's one. this is possibly due to the difference between the number of face-frames and the different detector.

5.3.2 *Testing on different multicropping configurations*

Having present the non-optimal crops of resized face-images, see section 4.2, eight multicropping configurations, for computing a single descriptor $\phi(I_t)$ from a face-image of the LFW dataset, were tested. These configurations were divided according to the scale, the bounding-box around the face, and the number of crops. Two size-configurations were used: a single scale of 256, and three scales $[256, 348, 512]$ (as the original testing procedure). The bounding-box of the face-images in LFW dataset is larger than face-images of VGG dataset, because of the different face-detectors used by

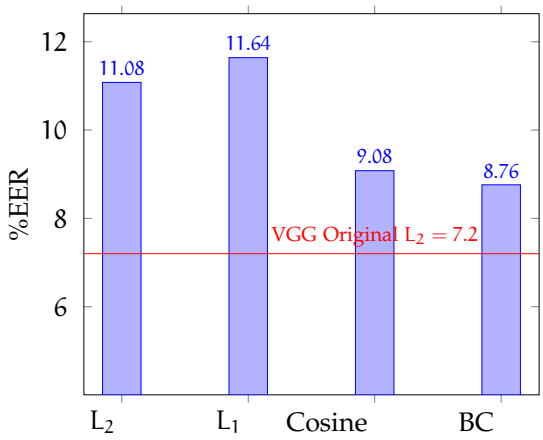
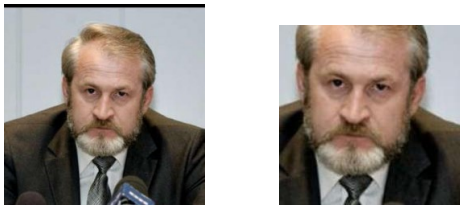


Figure 5.3.2: %EER vs similarity measurements for the unrestricted configuration of YTF dataset

their authors [GHLM07] [WHM11]. The authors in [PVZ15] utilized face-images that were detected on images of the LFW dataset using a DPM detector Figure 5.3.3, but as the Figure 4.2.1 showed previously, the crops of $[348, 521]$-resized face-images are not, in fact, faces. Therefore, utilizing a bounding-box, which possesses larger information of faces, could be beneficial for multicropping over three scales, because crops would be more likely to contain more face information. So, two-configurations of face-images were used: the original images, and the DPM-detected face-images, as DFR proposed see subsection 4.1.6, from LFW dataset.



(a) (b)

Figure 5.3.3: Original LFW face-image and a DPM-detected face-image

Moreover, the center cropping of three-resized face-images are more likely to be considered as faces, so one utilized, also, multicropping over the three scales but only considering the center of the face-images. The eight configurations are summarized in Table 5.3.1.

Name	Size	Face detection	$[224 \times 224 \times 3]$ Crops	# of crops
A	256	No	Four corners and center	10
B	256	No	Center	2
C	256	Yes	Four corners and center	10
D	256	Yes	Center	2
E	256, 384, 512	No	Four corners and center	30
F	256, 384, 512	No	Center	6
G*	256, 384, 512	Yes	Four corners and center	30
H	256, 384, 512	Yes	Center	6

Table 5.3.1: Multicropping configurations for LFW dataset testing

The Table 5.3.2 shows the %EER for all the multicropping configurations, and the similarity measurements for the LFW dataset testing. In general, the BC performs better than its counterparts; following the conclusion of [SF15], BC works better for higher-dimensionality. Besides, in comparison with the original testing configuration of DFR (the configuration G*[1]) with L_2, L1 and cosine measurements, the configurations, with a single scale (A,B,C,D) or with three scales using the original LFW face-images (E,F), show better results. This results are due to the using of suitable crops and face-images; since, the crops of all the scaled images would be considered as full-faces. This results are, also, consistent with the conclusion of [HZRS14], who stated that maintaining a complete content is important for improving the recognition's accuracy. However, the configuration H, which uses only center crops in all the scales, achieves the worst %EER; inclusively, the configuration H shows a deterioration of 6.860% (in L_2) with respect to the original configuration G* despite, intuitively, the more suitable crops taken from the center of images than the four corners, as the Figure 4.2.1 shows.

Figure 5.3.4 shows the relation between the %EER and the BC distance for all the configurations. Notice the superiority of the configurations of a single scale using DPM detection (C and D), and the configurations of three scales without DPM detection (E and F) with respect to the other configurations, especially, with respect to the original configuration G*. The best configuration (F) shows an improvement of 4.9%, 5.97%, and 4.77% compared to the original configuration (G*), the original configuration (G*) comparing with the L_2's %EER (red line), and the original publication's of DFR

1 Configuration G* refers to the original multicropping configuration defined by Alexnet [AK12]. It was thought for avoiding overfitting and for multisize robustness

	Similarity measurements			
Cropping conf	L_2	L_1	Cosine	BC
A	5.03	8.17	5.1	4.58
B	5.07	8.23	5.1	4.7
C	4.1	7.73	4.17	4.07
D	4.3	8.37	4.43	4.37
E	3.83	5.1	5.27	3.47
F	**2.77**	**4.7**	**2.73**	**2.4**
G*	8.37	8.03	11.9	7.3
H	15.23	12.87	11.6	11.43

Table 5.3.2: %EER for all the multicropping configurations, and the similarity measurements for the LFW dataset testing

(dark grey line) [PVZ15] respectively. Besides, comparing with the state of the art, this configuration is, also, superior to Fisher Vector Faces's %EER = 6.9 [SVZ14] and DeepFace's %EER = 2.65 [TYRW14], and comparable with the Fusion's %EER = 1.63 [YTW15].

5.3.3 *Training a CNN with SPP layer*

As discussed in chapter 4, a CNN with a 2-level SPP layer replacing the last max-pooling layer was trained. For that end, and utilizing the original CNN architecture of DFR (the A architecture in Table 3.2.2 subsection 3.2.2) as the initial state, the last three fully-connected layers, using batch GD with momentum $\gamma_1 = 0.9$, weight decay $\gamma_2 = 0.0005$ and three learning rates $\gamma_3 = [10^{-2}, 10^{-3}, 10^{-4}]$, were fine-tuned. Besides, the original CNN performs well for non-normalized face-images. However, in the literature (e.g. Alexnet and VGG), authors have proposed using mean-normalized images for stabilizing the optimisation algorithm. Therefore, the above CNN with an SPP layer is trained using mean-normalized face-images. During training, random-cropped, and -horizontal flipped crops of size $[224 \times 224 \times 3]$ over 256 resized mean-normalized face-images, see Figure 5.3.5, from the VGG dataset, see subsection 5.1.1, were used for training the CNN. Later, this CNN is referred as CNN-SPP network.

The CNN-SPP network was tested on the restricted configurations of LFW and YTF datasets. For the LFW dataset, specifically, all the multicropping configurations were tested. Table 5.3.3 shows the %EER for all the configurations, and the similarity mea-

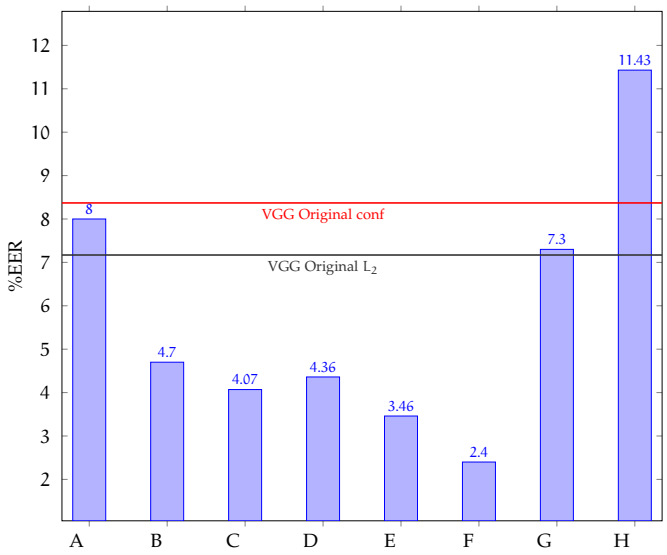
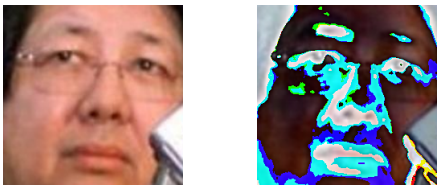


Figure 5.3.4: EER% vs BC distance for LFW dataset testing



(a) (b)

Figure 5.3.5: Mean-normalized face-image

surements. Contrary to the results of multicropping using the A-CNN subsection 5.3.2, the L_2 performs slightly better than the BC measurement; however, the %EER of these two measurements are better than the L1's, and cosine's ones.

Figure 5.3.6 shows the relation %EER and all the cropping configurations for L_2 similarity measurement. Similar to the original CNN's performance, the configurations of a single scale using DPM detection (C and D), and three scales without DPM detection (E and F) are superior with respect to the other configurations. The best configuration (F) shows an improvement of 4.53%, 4.8%, and 3.6% better than the original configuration (G), the original configuration (G) with D-CNN network (red line), and the original publication's of DFR (dark grey line) [PVZ15] respectively.

Nevertheless, this configuration (F) shows a deterioration of 1.17% with respect to the L_2's %EER of its similar configuration using the original CNN, see Figure 5.3.4.

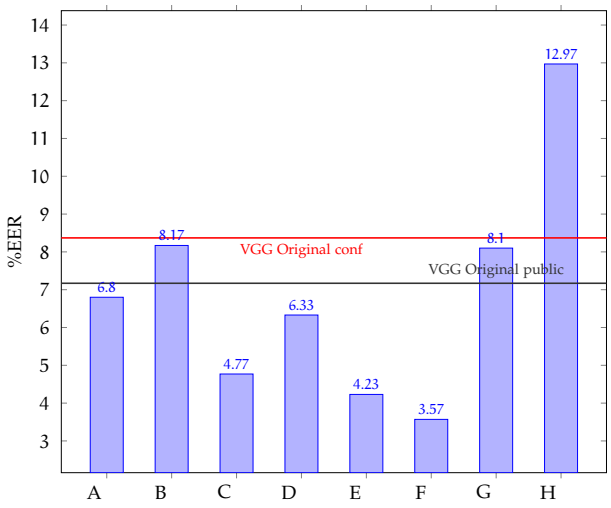


Figure 5.3.6: %EER vs L_2 distance for all multicropping configurations on LFW dataset using the CNN-SPP network

The %EER of configurations (C,D,E, and F) from both networks, the original CNN and the CNN-SPP, are superior to the one of the original multicropping configuration (G), and the original publication %EER itself. These results confirm that multicropping over different scales is not beneficial for computing a single face-descriptor, because crops of scaled-images are not faces—one can say that crops portray scaled-features of faces, e.g. a single eye, or a single ear, or parts of the mouth, but they do not posses enough information of a complete face. The best configuration (F) uses crops from the center of the face-images (original LFW dataset) in all the scales. As the bounding-box of the original LFW dataset covers a bigger area of the head (not the face), center crops of scaled face-images contain still a complete face. Besides, utilizing the configuration (F), the computation of a single face-descriptor is computationally cheaper than using the configuration (G), since one must compute five times lesser number of descriptors. In addition, multicropping configurations of a single scale using the DPM detector perform, also for both networks, better compared to the original configuration (G).

For the testing on YTF dataset, the Figure 5.3.7 shows the relation between the %EER vs the four similarity measurements, the original CNN's %EER, see Figure 5.3.2, (red line), and the original publication's one (dark grey line). The BC's %EER of the CNN-SPP shows a relative deterioration of 0.48% and 2.04% with respect to the

original CNN's and the original publications's respectively. The BC distance shows a superior performance compared to its counterparts, being consistent with the previous results.

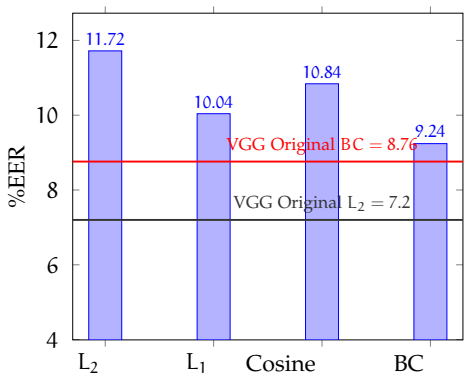


Figure 5.3.7: %EER vs similarity measurements for the restricted configuration of YTF dataset using the CNN-SPP as descriptor extractor

5.3.4 *Testing on half-occluded face-images*

As discussed in section 4.2, the performance of the previous CNNs (the original CNN, CNN-SPP) was tested using extreme-occluded face-images. Pointedly, half of the face-images, one face-image per pair or non-pair in the restricted configuration of LFW dataset, were artificially and randomly occluded. The face-images of LFW dataset, detected with the DPM, are, in general, face-centered and frontal-oriented. So, a half-occluded face-image contains, approximately, a half of a face, see Figure 5.3.8. For each pair or non-pair, a descriptor $\phi(I_{t_{half}})$ of half-face image is compared with a descriptor $\phi(I_t)$ of a face-image.

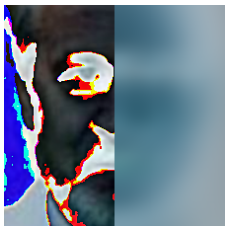


Figure 5.3.8: Example of a half-occluded face-image

For the original CNN, the %EER for the multicropping configurations (A,C, and F) and all the similarity measurements are shown in Table 5.3.4. Notice the increment of the %EER with respect to the experiments with entire face-images (non-occluded face-images). Specially, the multicropping configurations using three sizes show a high %EER; in fact, the performance becomes random, when using the multicropping configurations (E, G, and H). This is due to the use of black crops, which do not have any relevant information. Configuration (C) shows better results than their counterparts (configuration A without DPM detection, and multisize configuration F). Besides, and contrary to previous results, the Cosine distance shows, slightly, better performance than the other similarity measurements. Compared to the original results, see Table 5.3.2, the %EER of this testing setting presents a large deterioration, e.g. there is an absolute difference of 8.03% between the configuration (D)'s %EER using the BC similarity for both testing settings.

For testing the CNN-SPP using half-occluded face-images, only the multicropping configuration (C) was used. The Figure 5.3.9 shows the %EER vs similarity measurements, including, the the best %EER when using the original CNN as descriptor extractor (red line), see Table 5.3.4. The performance of the BC distance shows a superior performance with respect to the other distances, following the previous results. However, it is not superior to the original CNN's one, having an absolute difference of 1.43%. This results show that there is not a benefit of using an SPP layer for verification of extreme-occlusion of faces.

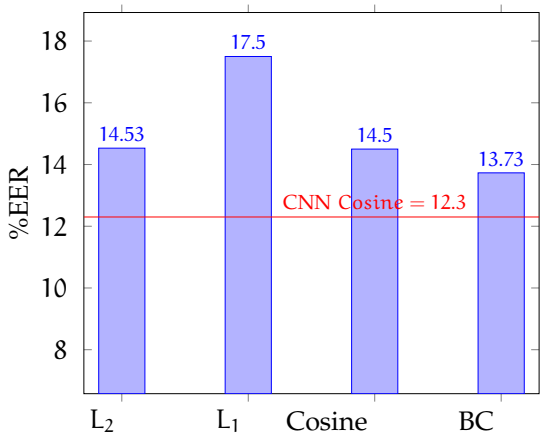


Figure 5.3.9: %EER vs similarity measurements for the CNN-SPP using half-occluded face-images of LFW dataset

	Similarity measurements			
Cropping conf	L_2	L_1	Cosine	BC
A	6.8	7.7	8.97	7.53
B	8.17	9.93	9	8.6
C	4.77	5.47	6.13	5.13
D	6.33	6.9	6.63	6.13
E	4.23	4.73	8.6	5.27
F	**3.57**	**4.6**	**5.27**	**4.2**
G*	8.1	7.8	14.6	9.1
H	12.97	10.6	15.27	12.33

Table 5.3.3: %EER for all the multicropping configurations, and the similarity measurements using the CNN-SPP network on the LFW dataset testing

Occlusion	Cropping conf	Similarity measurements			
		L_2	L_1	Cosine	BC
Yes	A	17.23	18.07	16.57	16.13
Yes	**C**	**13.23**	**14.43**	**12.3**	**12.57**
Yes	F	29.3	21.17	21.47	21.43
No	D	6.33	6.9	6.63	6.13

Table 5.3.4: %EER for all the multicropping configurations, and the similarity measurements using the CNN network on the LFW dataset testing

5.3.5 *Half-max pooling on the last convolutional layer's output*

Analysing the feature-maps of original face-images and half-occluded face-images through the convolutional layers, see Figure 5.3.10, half of the feature-maps of the convolutional layers are not, for the earlier convolutional layers, or are partially, for the last convolutional layers, activated. Therefore, a sort of half (left-right sides)-max pooling on the fifth convolutional layer's feature-map is utilized, such that the largest activations from one of the sides of the feature-maps are copied to the other side—under the assumption that faces are centered and not tilted or rolled—, producing full-face feature-maps.

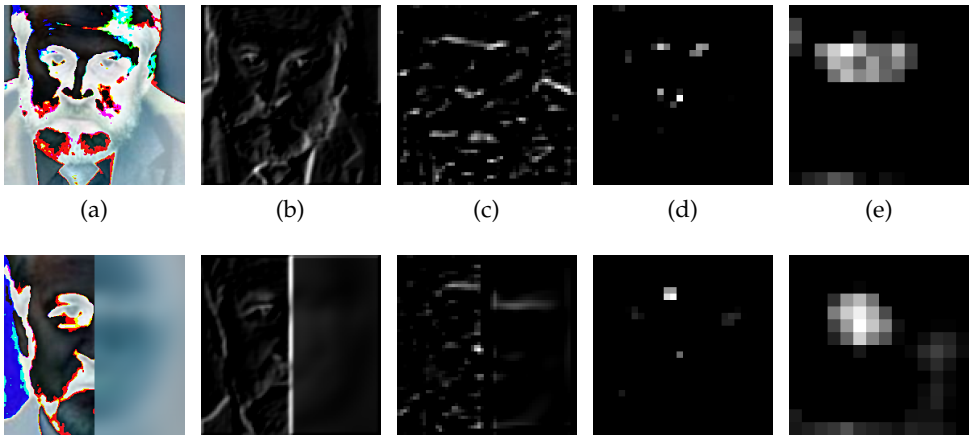


Figure 5.3.10: The second (b,g), third (c,h), forth (d,i), and fifth (e,j) convolutional layer's feature-maps from a original face-image (a) and a half-occluded face-image (f) of LFW dataset

Figure 5.3.11 shows the %EER vs the CNN and CNN-SPP with the half-max-pooling on top of the fifth convolutional layer for the multicropping configuration (C) on the non-occluded and side-random-occluded face-images of the LFW dataset, comparing, also, with the previous CNN's and CNN-SPP's %EER on side-random-occluded face-images, see Table 5.3.4 and Figure 5.3.9, and on non-occluded face-images, see Figure 5.3.4 and Table 5.3.3, (brown and dark gray colours) of LFW dataset. Comparing the %EER of both networks when applying a half-max pooling (blue and red columns) on Side-random-occluded face-images, the CNN-SPP shows an absolute improvement of 2.57%. However, there is no improvement, and deterioration of 0.67% with respect to the same network without the half-max pooling on the side-random-occluded and

the non-occluded face-images (red and grey columns) respectively. In the case of the CNN with half-max pooling, there is an absolute deterioration of 3.73% and 1.71% on the performance compared to the same network without the half-max pooling on side-random-occluded and the non-occluded face-images (blue and brown columns) correspondingly. Nonetheless, the %EER for both networks with half-max pooling on non-occluded face-images is, relatively, 19.1% superior than the original publication's (red line).

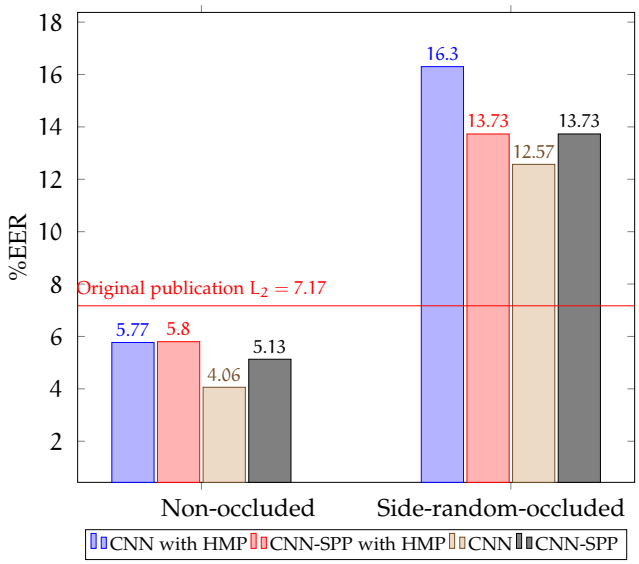


Figure 5.3.11: %EER vs CNN and CNN-SPP performance applying the half-max pooling on top of the fifth convolutional layer on the non-occluded and side-random-occluded face-images of LFW dataset using the BC distance for multicropping configuration (C)

5.3.6 *Mirroring the last convolutional layer's output*

For observing whether parts of feature-maps in the last convolutional layer, which correspond to occluded areas from the face-image, are activated and their values are higher than the other side's, only the left side of face-images were occluded, and, instead of implementing a half-max pooling—in which case, the left side of the last convolutional layer's feature maps with non-face activations could affect the right side—, a mirroring from right to left side of the last convolutional layer's feature map is utilized, assuming that the right part of feature maps should be activated

and the left side should not. A testing with both networks (CNN, and CNN-SPP), mirroring (CNN-mirror and CNN-SPP-mirror) and not mirroring (CNN and CNN-SPP) their last convolutional layer's feature maps, and using the configuration (C) for left-side-occluded was performed, which results using the BC distance as similarity measurement are summarized in Figure 5.3.12. Despite the absolute reduction of 1.27% between the %EER of the CNN-SPP and the same network applying the mirroring, applying a mirroring of, as well as, a half-max pooling on, see previous experiment Figure 5.3.11, the last convolutional layer's feature-maps do not provide any benefit with respect to face verification task for extreme-occluded face-images; inferring that, filters (neurons) in convolutional layers do not activate symmetrically, e.g. a neuron can activate only when it "sees" an eye, but not necessarily for both eyes. Besides and even though frontal faces are, in general, symmetric, the spatial relation of facial-features, in specific in convolutional feature-maps, are not. Therefore, implementing a mirroring, trying to copy features from one side (full of important information of faces) to another (with null or weak information of faces), to fill a complete feature-map does not improve the performance on extreme-occluded faces.

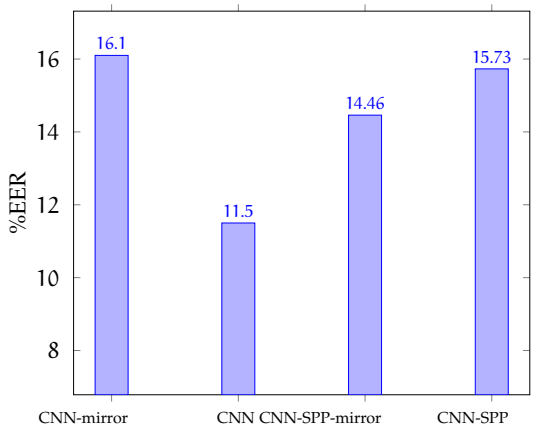


Figure 5.3.12: %EER vs CNN and CNN-SPP performance when mirroring and not mirroring the last convolutional layer's feature-maps on the left-occluded face-images of LFW dataset using the BC distance for multicropping configuration (C)

5.3.7 *Mirroring the input images*

As seen in the previous experiments on extreme-occluded face-images, the approaches for copying meaningful information from strongly activated parts of feature-maps

were not successful. Neurons do not activate for symmetrical features, making non-viable mirroring important information for filling feature-maps that are required for an appropriate descriptor extraction. Nonetheless, these approaches do not fully deteriorate the performance of deep CNNs on non-occluded face-images, see Figure 5.3.11. For that reason and for comparison purposes, non-occluded parts of face-images were mirrored before being fed to the CNNs, see Figure 5.3.13, in such a way that, early neurons (from the first convolutional layers) activate for facial-features from full-faces images.

(a) (b)

Figure 5.3.13: Mean-normalized face-image

Table 5.3.5 summarizes the %EER vs all the similarity measurements from all the one-size multicropping configurations (A,B,C and D) for the original CNN, and the configuration C* from the CNN-SPP. For all these multicropping configurations, the %EER shows a significant improvement with respect to the same networks, when feeding extreme-occluded face-images, see Table 5.3.4, and Figure 5.3.9. Besides, the CNN under this, specifically, type of inputs shows a relative improvement of 21.2% better than the CNN-SPP. Being consistent with previous results, using the BC distance as similarity measurement boosts the performan of the CNNs.

Figure 5.3.14 shows, in specific, the BC distance for both networks, when feeding the occluded and mirrored face-images, including also, the %ERR of the networks when applying the half-max pooling and the mirroring on the last convolutional layer's feature map. Notice that using mirrored face-images helps to overcome the problem of extreme-occluded faces; concretely, there is an absolute improvement of 2.91% between the CNN's %ERR. However, this approach is 34.1% inferior compared to the original publication's %EER (on non occluded face-images).

The approaches for facing extreme-occlusion in face-images depend, strongly, on the face-alignment of the testing dataset. The LFW and YTF datasets contain, in general, frontal faces. Using an additional face-alignment might boost the performance of previous approaches (half-max pooling on-, mirroring of last convolutional layer's

	Similarity measurements			
Cropping conf	L_2	L_1	Cosine	BC
A	10.23	14.73	10.1	10
B	10	14.8	10.23	10.2
C	**10.26**	**15.5**	**10**	**9.66**
D	10.366	16.46	10.2	10
C*	12.533	12.37	12.8	12.26

Table 5.3.5: %EER for all the one size multicropping configurations, and the similarity measurements using the CNN and the CNN-SPP (C*) networks on mirrored face-images of the LFW dataset

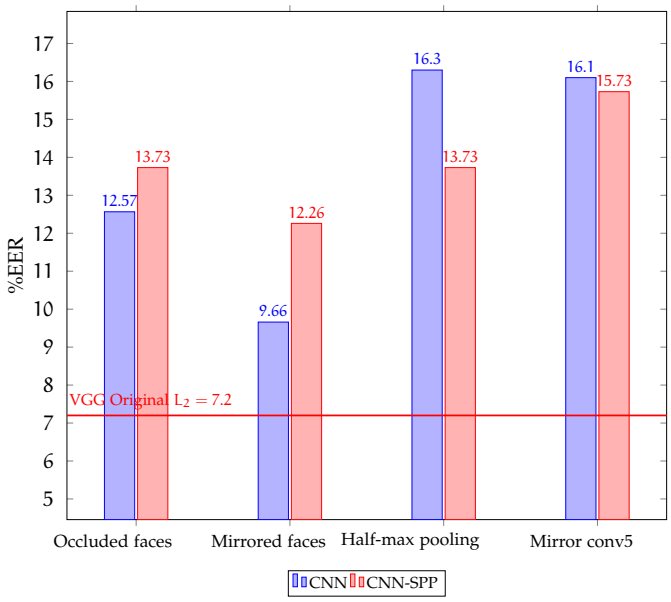


Figure 5.3.14: %EER vs CNN and CNN-SPP performance when feeding extreme-occluded and mirrored face-images of the LFW dataset using the BC distance for multicropping configuration (C)

feature-maps, and mirroring of input images), since facial-features would be better positioned.

5.3.8 *Training triplet loss*

Following the approach in subsection 4.3.2, an additional fully-connected layer for each of the two networks, minimizing the the Empirical Triplet Loss, see subsection 4.1.4, using non- and half-occluded face-images as anchors, is fine-tuned. For that end, and following the triplet selection procedure of [FS15], a set of "hard triplets" T from the testing LFW dataset (following the unrestricted configuration, see subsection 5.1.2) was computed, such that they ensure fast convergence on training. For a half-occluded face-image as an anchor a, a triplet $[a, p, n] \in T$ is composed of a hard positive $p = argmax_{p_i} \|X_a - X_{p_i}\|_2^2$ from a mini-batch of positives images, and a hard negative $n = argmin_{n_i} \|X_a - X_{n_i}\|_2^2$ from a mini-batch of negative images. The positive and negative mini-batches are built using images from the training set of the unrestricted configuration of LFW dataset—none of the face-images from the 6000 pairs of the testing dataset are used for learning the embeddings. Besides, a set of suitable triplets, which violate the condition $\|X_a - X_n\|_2^2 > \|X_a - X_p\|_2^2$, are extracted from the set of hard triplets. For the CNN-SPP network, an additional layer embedding the non-occluded face-images, using batch GD with momentum $\gamma_1 = 0.9$, weight decay $\gamma_2 = 0.0005$ and learning rate $\gamma_3 = 10^{-3}$ with a batch's size of 64 for 100 iterations; and an additional layer embedding the half-occluded face-images, using the same aforementioned parameters but with a batch's size of 32, were independently learnt. As a result, the CNN-SPP with an embedding computes a more distinctive and compact 1024-dimensional face-descriptors X_t.

The experiments using the CNN-SPP for both embeddings, multicropping configuration (C) on non- and extreme-occluded face-images were executed. Their %EER using the L_2 distance are summarized in Figure 5.3.15, including the previous results using the CNN (blue bar) and the CNN-SPP (brown bar) on the same input sets. Notice that the CNN with embedding the non-occluded faces show a small deterioration of 0.5% with respect to the CNN no T.L (triplet loss) in the experiments with non-occluded. Besides, the CNN-SPP with embedding the non-occluded and the extreme-occluded faces show, respectively, a small absolute improvement of 0.34% and a small deterioration of 0.4% compared to the CNN-SPP no T.L in the experiments with non-occluded images. However, for experiments with extreme-occluded faces, there is a deterioration of the performance for all the embeddings, showing no benefit for face-verification on extreme-occlusion. Thus, and contrary to the original publication's conclusions

[PVZ15], the embeddings do not boost, significantly, the performance of the deep CNN; indeed, the most remarkable benefit of the embeddings is the dimensionality reduction of the D−dimensional face-descriptor $\phi(I_t)$, see subsection 4.1.2. Besides, the layer implementing the Embedding maps the face-descriptors into a Euclidean space, so the BC distance produces random results.

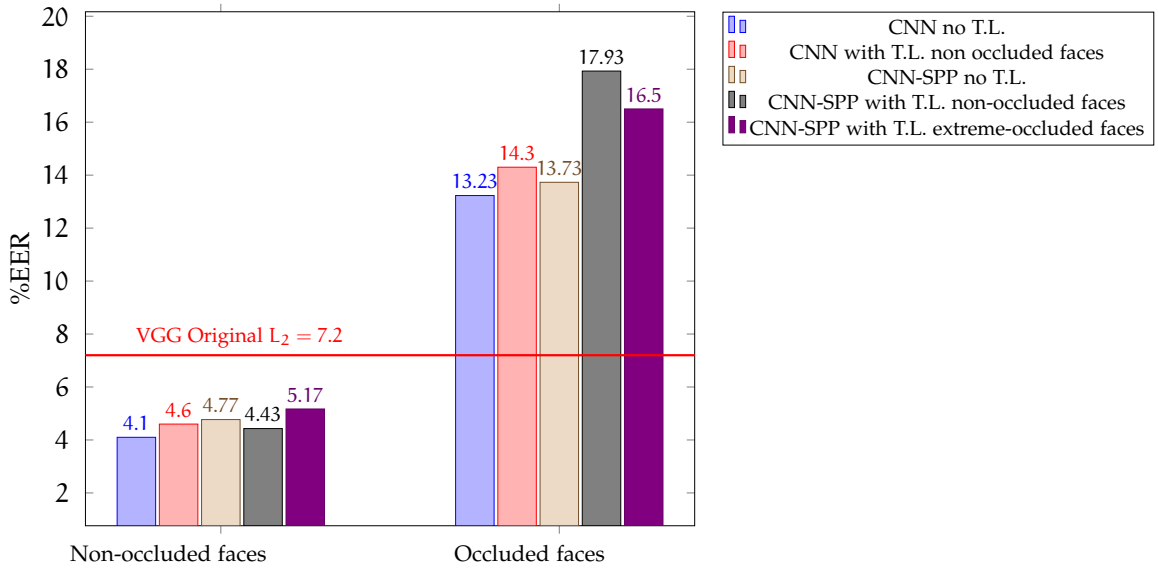


Figure 5.3.15: %EER vs CNN-SPP with and without the two embeddings performance when feeding non- and extreme-occluded face-images of the LFW dataset using the L_2 distance for multicropping configuration (C)

5.3.9 *Testing different occlusion proportions*

Until now, experiments with non- and extreme-occlusion cases have been presented. The performance of the previous networks (the original CNN and CNN-SPP), and the approaches for facing occlusion (half-max pooling and mirroring of the last convolutional layer's feature-maps) on different proportions of occlusions are evaluated. Portions of face-images, concretely, [5%, 10%, 15%, 20%, 30%, 40%, 50%] are occluded at random. Table 5.3.6 shows the BC's %EER for both networks, and the approaches for facing occlusion from all the occlusion proportions. Consistent to the previous results on comparisons among similarity distances, the BC distance shows better performance, being suitable for face-verification. The original CNN is superior, for all the proportions, compared to the CNN-SPP with and without the half-max pooling or mirroring

the fifth convolutional layer's feature-maps. Comparing with the original publication's %EER = 7.17 for non-occluded face-images, the performances of the CNN, the CNN with SPP, and the approaches for facing partially- and extreme-occlusion are remarkable for proportions until to 20%. They are not superior to the original %EER, but they present a small deterioration. The worst case (50% of occlusion) using CNN-SPP with mirroring the fifth convolutional layer's feature-maps shows a EER% equal to the double of the original publication's one.

	Occlusion proportions						
Network	5%	10%	15%	20%	30%	40%	50%
Original CNN	**4.93**	**5.6**	**5.97**	**7.47**	**8.37**	**8.9**	**11.06**
CNN-SPP	6.96	7.27	8.57	10.03	11.43	11.4	13.73
CNN-SPP with half-max-pool	7.07	8.5	9.3	11.23	12.5	12.9	13.73
CNN-SPP with mirroring the conv5	6.4	7.53	8.73	10.53	11.7	11.8	14.26

Table 5.3.6: %EER of the networks and approaches for facing occlusion using multicropping configuration C, and the BC similarity measurement on proportional occluded face-images of the LFW dataset

6 CONCLUSIONS

The thesis goal is to analyze the performance of deep CNN architectures for face-verification, when faces are non-, partial- and extremely-occluded, including the evaluation of different similarity measurements, variations of the CNN's architecture or CNN's feature maps, and a validation of testing procedures, commonly, used in object recognition and verification.

A validation and a discussion of the testing procedure, concretely, the use of several crops extracted from corners and center of three scaled face-images that was introduced in [AK12] and followed by [SZ15] for general object recognition and [PVZ15] for face-recognition and verification, who also utilized the Deformable Parts Model face-detector for improving the bounding-box around faces, is presented. Contrary to [AK12], [SZ15], and [PVZ15] results, but consistent to [HZRS14], the use of multiple crops over three scales is not beneficial for face-recognition and verification purposes, because crops, mainly from the corners, of the largest scales do not contain meaningful information of faces, affected, clearly, by the bounding-box's size in images, which does not allow to use entire faces; that is, including also the head contour. After analyzing different combinations for extracting suitable crops, and a validation whether using or not a face-detector, which changes the bounding-box around faces, it is found that configurations using a single scale including a face-detector, or configurations using three scales from only the center part of images without a face-detector—or with a bounding-box around faces that covers also the head—, are, certainly, superior—and computationally cheaper since lesser number of crops are fed to the deep CNNs—compared to the formulated configuration in the literature. This finding allows to partially conclude that it is necessary to analyze whether the testing procedures of general object recognition from related works are suitable to conduit a proper testing on a particularly object recognition and verification; in this case, face-verification.

Additionally, four distance measurements as similarity metric for face verification were compared. Being consistent with [Blo81] and [SF15], the Bray-Curtis Dissimilarity, in general, improves the performance for solving the task of face-verification when comparing high-dimensional face-descriptors computed by deep CNNs; except,when the Embedding triplet loss is used, in which case the final descriptors are mapped to a Euclidean space, and the L_2 distance performs better.

Moreover, an evaluation of performing face-verification, on non-, extreme- and partially-occluded face-images using two similar deep CNNs—the two deep CNNs vary, only, on the last pooling layer, specifically, one of them uses a Spatial Pyramid Pooling layer (SPP) instead—, was also presented. For the CNN without an SPP layer, it was found that it achieves an %EER of 2.4% when using non-occluded face-images of the LFW dataset—and clearly using a proper multicropping configuration—, being comparable to the state of the art, and with the DFR's %EER = 1.05 [PVZ15] that uses the Embedding minimizing the triplet Loss, even though the deep CNN in question does not utilize the Embedding. In addition, its performance does not extremely deteriorate from partial %EER = 4.93 up to the extreme-occlusion %EER = 11.06% conditions. However and contrary to [HZRS14], the deep CNN with an SPP layer does not overpasses the original one, but this deep CNN is, also, not extremely affected by partial %EER = 6.96% up to the extreme-occlusion %EER = 13.73% conditions. Based on these observations in occlusion, it can be partially concluded that deep CNNs can verify whether two face-images portray the same identity; inclusively, when the faces are partially- up to half- occluded; that is, they can serve from information from parts of faces, as humans perception does. Besides, two approaches for boosting the performance of the CNN with an SPP layer, which manipulate directly the last convolutional layer's feature-maps for filling possible non-activations caused by occluded areas in faces with their opposite side activations, were also introduced. Nevertheless, these two approaches did not improve the performance. Since, neurons do not necessarily activate to symmetrical facial-features on images, and the face-images in the testing datasets are not completely symmetric. Also and following the Embedding procedure of [PVZ15], an additional fully-connected layer for both deep CNNs, implementing an embedding using a triplet loss as the objective function for the optimization process [FS15], was utilized. Although the Embedding strongly improved the performance of the network in [PVZ15], it did not boost the performance of the here utilized deep CNNs for non- and extreme-occluded face-images; but, it reduces the dimensionality of the final descriptor. As future work, the deep CNN with SPP can be trained from scratch, such as the neurons in convolutional layers adapt also according to the influence of the SPP layer on the network; besides, an additional face-alignment can be utilized, such that it boost the two aforementioned approaches for facing occlusion.

BIBLIOGRAPHY

[AHP06] Ahonen, Timo ; Hadid, Abdenour ; Pietikainen, Matti: Face description with local binary patterns: Application to face recognition. In: *IEEE transactions on pattern analysis and machine intelligence* 28 (2006), Nr. 12, S. 2037–2041

[AK12] Alex Krizhevsky, Geoffrey E H. Ilya Sutskever S. Ilya Sutskever: ImageNet Classification with Deep Convolutional Neural Networks. (2012). http://dx.doi.org/{kriz,ilya,hinton}@cs.utoronto.ca. – DOI kriz,ilya,hinton@cs.utoronto.ca

[AR15] Antonio Rama, Francesc T.: Un nuevo método para la detección de caras basado en Integrales Difusas. (2015). http://dx.doi.org/{alrama,tarres}@gps.tsc.upc.edu. – DOI alrama,tarres@gps.tsc.upc.edu

[BC57] Bray, J R. ; Curtis, John T.: An ordination of the upland forest communities of southern Wisconsin. In: *Ecological monographs* 27 (1957), Nr. 4, S. 325–349

[BHK97] Belhumeur, P. N. ; Hespanha, J. P. ; Kriegman, D. J.: Eigenfaces vs. fisherfaces: Recognition using class specific linear projection. In: *IEEE Transactions on pattern analysis and machine intelligence* 19 (1997), Nr. 7, S. 711–720

[Blo81] Bloom, S. A.: Similarity Indices in Community Studies Potential Pitfalls. In: *Marine Ecology- Progress Series, vol.5, 125-128* (1981)

[CC01] C. C, A. Hinnerburg A. A. K. Aggarwal: On the Surprising Behaviour of Distance Metrics in High Dimensional Space. In: *ICDT, pp 420-434* (2001)

[CEL87] Craw, Ian ; Ellis, H ; Lishman, J R.: Automatic extraction of face-features. In: *Pattern recognition letters* 5 (1987), Nr. 2, S. 183–187

[doc16] documentation, OpenCV: *Face Recognition with OpenCV*. url-http://docs.opencv.org/2.4/modules/contrib/doc/facerec, 2016

[FFL16] Fei-Fei Ly, Justin J. Andrej Karpathy K. Andrej Karpathy: *CS231n: Convolutional Neural Networks for Visual Recognition*. url-http://cs231n.stanford.edu/, 2016

[Flooo] Floyd, Thomas L.: *Fundamentos de Sistemas Digitales*. Pearson Education, 2000. – 7th ed.

[FM82] Fukushima, Kunihiko ; Miyake, Sei: Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. In: *Pattern recognition* 15 (1982), Nr. 6, S. 455–469

[FS15] Florian Schroff, James P. Dmitry Kalenichenko K. Dmitry Kalenichenko: FaceNet, A unified Embedding for Face Recognition and clustering. (2015). http://dx.doi.org/{fschroff,dkalenichenko,jphilbin}@gmail.com. – DOI fschroff,dkalenichenko,jphilbin@gmail.com

[GBB11] Glorot, Xavier ; Bordes, Antoine ; Bengio, Yoshua: Deep Sparse Rectifier Neural Networks. In: *Aistats* Bd. 15, 2011, S. 275

[GEAR13] Giot, Romain ; El-Abed, Mohamad ; Rosenberger, Christophe: Fast computation of the performance evaluation of biometric systems: Application to multibiometrics. In: *Future Generation Computer Systems* 29 (2013), Nr. 3, S. 788–799

[GHLM07] G.B Huang, T. B. M. Ramesh R. M. Ramesh ; Learned-Miller, E.: Labeled faces in wild: A database for studying face recognition in unconstrained environments. In: *Technical report 07-49, University of Massachusetts* (2007)

[Hai14] Haines, Duane E.: *Principios de Neurociencia: Aplicaciones básicas y clínicas*. Elsevier Saunders, 2014. – 2014 ed.

[HL01] Hjelmas, E. ; Low, B. K.: Face Detection: Survey. In: *Computer Vision and Image Understanding, vol. 83, no. 3, pp. 236-274* (2001)

[HZRS14] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Spatial pyramid pooling in deep convolutional networks for visual recognition. In: *European Conference on Computer Vision* Springer, 2014, S. 346–361

[Kan73] Kanade, T: Picture processing system by computer complex and recognition of human faces. In: *Doctoral dissertation, Kyoto University* 3952 (1973), S. 83–97

[KBBN09] Kumar, N ; Berg, A. C. ; Belhumeur, P. N. ; Nayar, S. K.: Attribute and simile classifiers for face verification. In: *2009 IEEE 12th International Conference on Computer Vision* IEEE, 2009, S. 365–372

[Kon05] Konar, A: *Computational Intelligence: Principles, Techniques, and Applications*. Springer Science & Business Media, 2005. – 2005 ed.

[LCDH+90] Le Cun, B B. ; Denker, John S. ; Henderson, D ; Howard, Richard E. ; Hubbard, W ; Jackel, Lawrence D.: Handwritten digit recognition with a back-propagation network. In: *Advances in neural information processing systems* Citeseer, 1990

[LDS+89] LeCun, Yann ; Denker, John S. ; Solla, Sara A. ; Howard, Richard E. ; Jackel, Lawrence D.: Optimal brain damage. In: *NIPs* Bd. 2, 1989, S. 598–605

[LKF+10] LeCun, Yann ; Kavukcuoglu, Koray ; Farabet, Clément u. a.: Convolutional networks and applications in vision. In: *ISCAS*, 2010, S. 253–256

[LSP06] Lazebnik, Svetlana ; Schmid, Cordelia ; Ponce, Jean: Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)* Bd. 2 IEEE, 2006, S. 2169–2178

[MBPVG14] Mathias, M ; Benenson, R ; Pedersoli, M ; Van Gool, L: Face detection without bells and whistles. In: *European Conference on Computer Vision* Springer, 2014, S. 720–735

[OPM02] Ojala, Timo ; Pietikainen, Matti ; Maenpaa, Topi: Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. In: *IEEE Transactions on pattern analysis and machine intelligence* 24 (2002), Nr. 7, S. 971–987

[PVZ15] Parkhi, Omkar M. ; Vedaldi, Andrea ; Zisserman, Andrew: Deep face recognition. In: *British Machine Vision Conference* Bd. 1, 2015, S. 6

[RDS+15] Russakovsky, Olga ; Deng, Jia ; Su, Hao ; Krause, Jonathan ; Satheesh, Sanjeev ; Ma, Sean ; Huang, Zhiheng ; Karpathy, Andrej ; Khosla,

Aditya ; Bernstein, Michael u. a.: Imagenet large scale visual recognition challenge. In: *International Journal of Computer Vision* 115 (2015), Nr. 3, S. 211–252

[Roj96] Rojas, Raúl: *Neural networks: a systematic introduction*. Springer Science & Business Media, 1996

[SF15] Sudholt, Sebastian ; Fink, Gernot A.: A Modified Isomap Approach to Manifold Learning in Word Spotting. In: *German Conference on Pattern Recognition* Springer, 2015, S. 529–539

[SVZ14] Simonyan, Karen ; Vedaldi, Andrea ; Zisserman, Andrew: Learning local feature descriptors using convex optimisation. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36 (2014), Nr. 8, S. 1573–1585

[SZ15] Simonyan, Karen ; Zisserman, Andrew: Very Deep Convolutional Networks for Large-Scale Image Recognition. In: *ICLR* (2015). http://dx.doi.org/{karen,az}@robots.ox.ac.uk. – DOI karen,az@robots.ox.ac.uk

[TP91] Turk, Matthew ; Pentland, Alex: Eigenfaces for recognition. In: *Journal of cognitive neuroscience* 3 (1991), Nr. 1, S. 71–86

[TYRW14] Taigman, Yaniv ; Yang, Ming ; Ranzato, Marc'Aurelio ; Wolf, Lior: Deepface: Closing the gap to human-level performance in face verification. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, S. 1701–1708

[WHM11] Wolf, Lior ; Hassner, Tal ; Maoz, Itay: Face recognition in unconstrained videos with matched background similarity. In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on* IEEE, 2011, S. 529–534

[YKA02] Yang, M-H ; Kriegman, D J. ; Ahuja, N: Detecting faces in images: A survey. In: *IEEE Transactions on pattern analysis and machine intelligence* 24 (2002), Nr. 1, S. 34–58

[YTW15] Y. Taigman, M. R. M. Yang Y. M. Yang ; Wolf, L.: Web-scale training for face identification. In: *CVPR* (2015)