ISSN 1688-2806

Universidad de la República
Facultad de Ingeniería

A Study of Deep Learning and its applications to Face Recognition techniques

Tesis presentada a la Facultad de Ingeniería de la Universidad de la República por

Fernando Suzacq

en cumplimiento parcial de los requerimientos para la obtención del título de Magister en Ingeniería Eléctrica.

Directores de Tesis

Matías Di Martino	Universidad de la República & Duke University
Mauricio Delbracio	Universidad de la República

Tribunal

Alicia Fernández	Universidad de la República
José Lezama	Universidad de la República
Javier Preciozzi	Universidad de la República
Pablo Iturralde	Universidad Católica del Uruguay

Director Académico

Matías Di Martino	Universidad de la República & Duke University

Montevideo
domingo 1 agosto, 2021

A Study of Deep Learning and its applications to Face Recognition techniques, Fernando Suzacq.

ISSN 1688-2806

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 109 páginas.
Compilada el domingo 1 agosto, 2021.
`http://iie.fing.edu.uy/`

El secreto para que las cosas sean hechas está en hacerlas.

DANTE ALIGHIERI

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Realizar un posgrado no es tarea menor, como aprendí en estos años. Aún menos cuando el foco de atención es compartido con otros objetivos de igual o mayor importancia. Así que en primera instancia agradecerle a mis tutores, Matías Di Martino y Mauricio Delbracio, por permitirme recorrer este camino en condiciones atípicas. Realicé la maestría paralelamente a estar trabajando en tiempo completo, sumando otros momentos importantes profesional y personalmente. Con todo el impacto que esto puede tener en la dedicación e incluso hasta en la coordinación de simples reuniones, ellos siempre fueron muy comprensivos y flexibles para acomodarse a lo que podía brindar. Ámbos desde su lugar y en su momento me acompañaron y me enseñaron muchísimo. Solamente expresar mi más profunda admiración por su trabajo, conocimiento e inteligencia.

Particularmente agradecer a Matías por hospedarme en su casa en Durham, donde me sentí muy bien recibido. A Guillermo Sapiro por permitirme vivir la experiencia de trabajar temporalmente en un laboratorio de una de las universidades más prestigiosas del mundo. También a todo el equipo que trabaja allí. Tener acceso a dialogar con investigadores, ver distintas presentaciones, poder intercambiar con personas muy expertas en sus distintas áreas hicieron de esta experiencia mucho más enriquecedora para mi.

Agradecer en gran medida a la Universidad de la República y a la Facultad de Ingeniería por poder realizar esta maestría sin invertir más que mi tiempo. En un mundo extremadamente competitivo, donde en otros países obtienen posgrados en una cantidad de tiempo mucho menor y en un rubro altamente globalizado, es muy importante poder tener acceso a este tipo de programas. También, en particular al Instituto de Eléctrica por recibir a un emigrante del Instituto de Computación. Pienso que es muy bueno ver al InCo y al IIE trabajando más en conjunto en los últimos años. Ojalá esta tendencia siga.

Finalmente un agradecimiento enorme a mis personas más cercanas: Andi, mis padres y el resto de mi familia, mis amigos, mis socios de Pento, antiguos compañeros de trabajo y todas las otras personas que me acompañaron en estos años. En mayor o menor medida este esfuerzo los afectó a ellos también, ya sea por haberme perdido algún evento, compartir menos tiempo, tener que cubrirme en alguna oportunidad o apoyarme en estados de ánimo no tan disfrutables. Agradecerles porque siempre me brindaron fuerzas para seguir para adelante. Sin este apoyo, no lo podría haber hecho. En particular a mi esposa Andi, que fue quien me convenció de embarcarme en esta aventura. Hoy, cerca de terminar, si bien en distintas partes del camino no lo parecía, puedo ver que era la decisión correcta.

Esta página ha sido intencionalmente dejada en blanco.

A A. J. y A..

Esta página ha sido intencionalmente dejada en blanco.

Resumen

El siguiente trabajo es el resultado de la tesis de maestría de Fernando Suzacq. La tesis se centró alrededor de la investigación sobre el reconocimiento facial en 3D, sin la reconstrucción de la profundidad ni la utilización de modelos 3D genéricos. Esta investigación resultó en la escritura de un paper y su posterior publicación en IEEE Transactions on Pattern Analysis and Machine Intelligence.

Mediante el uso de iluminación activa, se mejora el reconocimiento facial en 2D y se lo hace más robusto a condiciones de baja iluminación o ataques de falsificación de identidad. La idea central del trabajo es la proyección de un patrón de luz de alta frecuencia sobre la cara de prueba. De la captura de esta imagen, nos es posible recuperar información real 3D, que se desprende de las deformaciones de este patrón, junto con una imagen 2D de la cara de prueba. Este proceso evita tener que lidiar con la difícil tarea de reconstrucción 3D. En el trabajo se presenta la teoría que fundamenta este proceso, se explica su construcción y se proveen los resultados de distintos experimentos realizados que sostienen su validez y utilidad.

Para el desarrollo de esta investigación, fue necesario el estudio de la teoría existente y una revisión del estado del arte en este problema particular. Parte del resultado de este trabajo se presenta también en este documento, como marco teórico sobre la publicación.

Esta página ha sido intencionalmente dejada en blanco.

Summary

The following work is the result of Fernando Suzacq's Master's Thesis. This thesis was focused on the investigation of 3D facial recognition, without reconstructing depth nor using generic 3D models. As a result of this investigation a paper was written that was later published in IEEE Transactions on Pattern Analysis and Machine Intelligence.

With the usage of active illumination, 2D facial recognition is improved and becomes more robust to low-light conditions or identity spoofing attacks. The main idea presented in this work is the projection of a high frequency light pattern on top of the test face. From a shot of this image, it is possible to decompose and recover real 3D information that becomes available from the deformations of the pattern according to the structure of the face, and also a 2D image of the test face. This process lets us avoid having to deal with 3D reconstruction which is a much harder task. In this work the theory behind this process is presented. Then a detailed explanation of how it was constructed is made and different experimental results are provided that proves its validity and usefulness.

For conducting this investigation, a previous study of the existing theory and a review of the State of the Art in this specific problem was necessary. Part of the result of this work is presented also in this document, as an introduction and theoretical background to the published paper.

Esta página ha sido intencionalmente dejada en blanco.

Prefacio

A partir de la siguiente sección, la tesis se encuentra en inglés. La razón de esto es que gran parte de la escritura se realizó en el contexto de la realización de artículos científicos y con colaboración de colegas extranjeros.

Fernando Suzacq

Esta página ha sido intencionalmente dejada en blanco.

Table of contents

Chapter 1

Introduction

The field of Artificial Intelligence has seen tremendous advancements over the past decades. It is a field rather young (partly because it is extremely tied to computing in general, which is also young), yet it has seen dramatic changes with an amazing speed, producing astonishing results. Today, we are all impacted by this field in some regard: be it a spam filter in our emails inboxes, a predictive keyboard in our cellphones, an automatic person tagging service in any social network.

The pervasiveness of these techniques in the latest years, coupled with some results seen in recent years, makes it a topic interesting to be studied in depth. This work was mainly centered on the application of machine learning techniques to images. Although ML is being applied to different kinds of data, e.g., structured data, text, time-series, images present some interesting challenges. On one hand, images convey a lot of information, can be complex to understand and in the last few years the amount of data generated and stored has increased dramatically. On the other hand, there are already a lot of mature techniques, previous to the advent of machine learning and particularly deep neural networks, that produced good results and have an interesting relationship with newer techniques. The field that tries to understand images programmatically and extract information and sense out of them is called Computer Vision.

Latest advancements of computer vision were due to breakthroughs in a specific Machine Learning technique: Neural Networks. Many discoveries were made in recent years, from figuring out how to train them properly and how to initialize them, to better architectures and techniques. Particularly, Convolutional Neural Networks (CNN), a specific NN architecture, are a great match for computer vision problems.

Machine Learning is a rather large body of knowledge. To gain some context and set the appropiate stage for what will come later, in the first part of this work we will try to summarize and explain some key ideas of the field. Focusing on deep learning, we will review different seminal works throughout the years that shaped this field into what it is today. We will try to explain some of the key components of neural networks as clearly and succinctly as possible. Then, to sum up this introduction to the field, we will move to view different architectures that have been proposed, specifically suited to be used in computer vision.

Computer Vision is comprised of a lot of subfields that tackle different problems. Some of them try to detect information present in the image such as objects, human poses, detecting and recognizing faces, detecting and recognizing written text, segmenting the area occupied by an object. Others deal with generating images: transferring style, generating objects from text or other input data.

There has been major advancements in the field of Face Recognition with the usage of these techniques. Accuracy results of face identification and verification in the latest datasets (which are composed of hundred of thousands of facial images) are astonishingly good. However, one shared characteristic of all of the proposed techinques is that they work with a 2D image taken from a 3D face. This presents some drawbacks. On one hand, some representational power is being lost. On the other hand, it is possible to confuse a verification system by presenting a 2D image to it, which it would have a hard time distinguishing it from a 3D face. We propose a method to recover real 3D information from a 2D image to mitigate this problem, by projecting a structured light pattern over the original 3D face. In the second part of this work, after presenting the problem of face recognition and the current architectures and solutions to it, we will present and explain this method in depth, both the theory in which it is based and experiments that have been made and show its usefulness in practice.

This work is ordered as follows: In the next Chapter we will discuss what is Machine Learning, and the different techniques of the field. Chapter 3 will focus on Face Recognition. We will present what this problem consists of and briefly approach related areas of study necessary for the next Chapter. Chapter 4 will focus on the proposed novel approach, resulting from the article: 3D face recognition without depth reconstruction nor usage of generic 3D models. Finally, in Chapter 5 we will provide the conclusions and lines of future work.

Chapter 2

Machine Learning

2.0.1. Definition & classifications

A Machine Learning algorithm is an algorithm that is able to learn from data [28]. There are a couple of commonly used definitions to describe what is learning, and one of them, presented by Mitchell in 1997 [64] is:

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Essentially, we can see that there are three basic components: **task**, **performance**, and **experience**. We need to be able to measure the performance of a task T with P, and get better at that task with experience E. There are a lot of different variations of T, P, and E, but informally, at the core of the definition, is the idea that the performance improves by "experience" and not by manual programming. Although the field is older, with other definitions tracing back to 1959, this definition is pretty clear and widely quoted. The main advantage that these algorithms have is that they can solve problems that may be too difficult to program, finding solutions that cannot be reached otherwise, automating and optimizing processes.

Tasks are usually defined depending on how the example at hand should be processed. An example can be thought of a particular instance of an experience E. Different tasks are: Classification (stating to which class an example belongs to), Regression (predict a numerical value given an example), Transcription/Translation (transform an example to another kind of representation - speech to text, spanish to english), synthesis and sampling (generate examples similar to those in the training data). This list is by no means exhaustive.

The performance measure is specific to the task that we are learning. For classification, the accuracy of the model is usually measured (right class - wrong class), while for regression a different measure has to be defined, since the error is a continuous value. The definition of the performance measure is extremely important, not only to compare two different algorithms, but also it will impact on how well and efficiently the algorithm will learn.

Finally, the way examples are experienced separates ML algorithms in broadly two groups: *unsupervised* and *supervised* learning.

Supervised learning algorithms deal with examples that have an associated target or label. For example, we know the correct class for each example in the training set in classification, or the correct translation in translation/transcription tasks. After training it, the algorithm will predict the corresponding label when given an unseen example. Algorithms are evaluated with a test set, that is, with a set of examples not seen before.

Unsupervised learning algorithms deal with examples that do not have an associated target. They experience them, and learn properties about their structure. They try to learn the entire probability distribution that generated a dataset. This can be later used for tasks like clustering or synthesis.

Both problems are not completely separate since an unsupervised learning problem of modelling $p(x)$ can be converted to n supervised problems by making use of the chain rule of probability in cases where we know the possible targets. This is, given $x \in R^N$ we learn $(x|x_i)$ for i in 1..N, where x_i are the targets. At the same time, we can solve the supervised problem of learning $p(y|x)$ by learning the joint distribution of $p(x, y)$ and then inferring $p(y|x) = \frac{p(x,y)}{\sum y' p(x,y')}$.

There are also other learning paradigms like semi-supervised learning (some examples include targets while other do not), multi-instance learning (a collection of examples is labeled as including an example of a class -or not- but individual examples are not identified), reinforcement learning (instead of experiencing a fixed dataset, the algorithm interacts with an environment and learns from its experience with it), and so on.

2.0.2. Algorithms

There are many different machine learning algorithms e.g., Linear regression, Support Vector Machines, Decision Trees, K-nearest neighbors, K-means clustering and so on. It is beyond the scope of this work to describe in detail these algorithms; there is abundant literature available addressing this issue, see for example, [7], [31], [65]. We will focus on a particular learning technique: Neural Networks. In particular, we will deal with Deep Learning, which is a specific architecture of Neural Networks, as we will see in the following Section.

Machine learning algorithms are very diverse, building on lots of different ideas. However, they share some generalities in the way they are trained and evaluated. Since they will be shared with Neural Networks as well, we will present these ideas in the following.

We can separate the process of working with machine learning models into two distinct stages: training and prediction. Our main goal when training a model is to be able to make predictions at a later time, but most importantly, on unseen data. That is what makes these algorithms so powerful: the ability to generate more information about an example in an automatic way, without intervention. Therefore, to mimic this situation, the available data X will always be split into two sets: X_{train} and X_{test}. We will use the first set to train our model, as its name implies, and we will use the second set **only** to evaluate the model's output, to have a similar situation to what the model would experience in a real scenario, and have a sense of how well it works. There are some subtle considerations that we must have when making this split, in order not to use data at training time that will not be available at prediction time (this is called *data leakage*).

Stating that an algorithm performs well (or not) depends on having a quantifiable definition of its performance. While it varies with the task at hand (e.g., classification vs. clustering), usually, a function is defined that conveys this value. For example, on a regression task (e.g., a model that tries to predict the price of a house), the Mean Squared Error can be used (the average of the squared error between the predictions and the ground truth values, for each example considered $MSE = \frac{1}{m}\sum_i(y_i^{pred} - y_i^{gt})^2$) or some other similar measure. If we are trying to do a cluster analysis, we may use the intra and inter-class euclidean distance of the clusters. In this way, we can compare the performance of two completely different models, since it only depends on the outputs.

Predictions will be a function of the inputs. For example, in a linear regression, this would be represented as $\hat{y} = w^T \cdot x$, but on other types of models this can be a lot more complex. Nevertheless, all of the algorithms share having a set of weights w that will determine the output for a given input x.

2.0.3. Learning process

In most algorithms, *learning* involves using the evaluation measurements (or related functions - *cost functions*) and optimization techniques to steer the outputs of the models towards the correct value to minimize or maximize the evaluation measurement. In practice, these optimization algorithms are used to tweak the weights of the model and get better predictions, therefore getting a better performance. When looking at neural networks, we will focus on a particular example, stochastic gradient descent. Despite its simplicity, it has been the most widely used optimization technique (or some variation of it), producing great results [28].

Most machine learning models also have settings that we can use to tune the behavior of the algorithm and achieve better results. These are usually called hyperparameters. For example, if we are trying to fit a curve through a set of points, the degree of the polynomial that fits the data would be a hyperparameter. They are not learned as part of the training algorithm and cannot be evaluated as part of the training set. In the example presented above, if we try to determine the degree of the polynomial by looking at the training data, we would end up with the highest value possible that fits all data points, while it might not be the best result. In order to deal with this problem, a validation set is usually defined: a hold-out set from the training set used to evaluate the different selections of hyperparameters.

When we train a machine learning model, we can compute an evaluation metric or error on the training set, called training error. However, as stated before, we want to be able to perform well on unseen inputs. Since the way we simulate unseen samples is with our test set, our most important metric will be the test error: the error on the test set. This is the key difference that separates these type of problems from optimization problems. As long as we keep the gap between the training error and the test error small, we can focus on minimizing the former. If the training error is considerably smaller than the test error, we will be facing with the problem of having overfitted the training set. The model can represent the training set well, but it does not generalize properly, which is seen on a poor test error. On the other hand, if the training error is large, we will say that the model is underfitting the training set. Often though, underfitting is an indication that the model lacks the complexity the data requires, meaning, is too simple to describe and learn the distribution of the data. This problem is usually related to the capacity of the model and its representation power. Balancing the learning process so that the learning algorithm fits the training set while at the same time generalizing properly is one of the central challenges in machine learning.

These considerations stated above are shared amongst most machine learning algorithms and set the general framework in which models are trained and evaluated. In the following Section, we will focus on Neural Networks and revisit some of the discussion introduced above. Having a concrete example might help clarify some concepts. At the same time we have to note that there are considerations that are particular to each method. This is what makes machine learning such an interesting and challenging field of study.

2.1. Deep Learning

2.1.1. History

Deep learning is a family of machine learning methods based on artificial neural networks. They learn the representation needed to perform a variety of tasks like classification or regression, replacing manual feature engineering.

It is useful to have some historical context of the development of deep learning, to understand how it has gotten to what it is today. Although it can be traced back to the 1940s, it was relatively unpopular until recent years, going through several name changes. The dramatic increase in available training data due to the ubiquity of computer systems and cheap storage solutions, together with the increase of computing power, has allowed the development of this field and enabled it to get astonishing results.

Artificial neural networks (ANNs) initially drew their inspiration from neurons in our brains, although lately, researchers have been trying to distance themselves from the analogy. They were introduced in 1943 and have a lot of landmark breakthroughs in their history. In this 1943 paper [63], the artificial neuron was proposed. It consists of one or more binary inputs and one binary output. The output activates when the number of activated inputs are above certain threshold. They can be used to build a network that can compute any logical proposition.

Artificial neurons used on nowadays networks where introduced by Rosenblatt in 1958 [74]. In his paper he introduced the Perceptron, one of the simplest ANN architectures. It is based on *linear threshold units* (LTUs), and its difference with

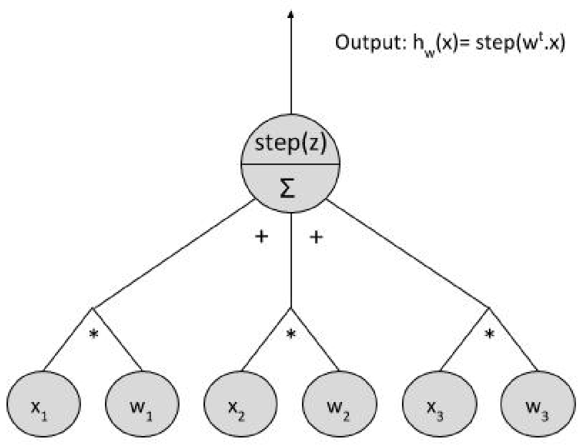


Figure 2.1: Diagram of an LTU. X_i correspond to the inputs and W_i to weights, one associated to each input. Each input is multiplied by its corresponding weight, and the resulting terms are summed up. Finally, the result of this sum is fed to a step function. Therefore, the output of the LTU corresponds to the application of the step function to the inner product of vectors X and W.

the artificial neuron previously explained is that inputs are numbers (instead of 0-1) and each input connection is associated with a weight. A weighted sum of inputs is performed and a *step* function is applied to that output. A diagram of its structure can be seen on Figure 2.1.

2.1.2. Perceptrons

An LTU can be used for a simple linear binary classification. Perceptrons, seen on Figure 2.2, have only one LTUs layer, where each unit is connected to every input. Inputs in general are represented by a pass-through neuron, that is, the outputs are the same as the input. Additionally, a neuron is added that represents bias, which always outputs 1.

To train a perceptron, examples of the training set are fed to the network one at a time, getting a prediction out of it. If the prediction is wrong, connections weights that would have led to a correct prediction are reinforced. This rule is summed up in the following equation:

$$w_{i,j}^{t+1} = w_{i,j}^{t} + \alpha(y_j^{gt} - y_j^{pred})x_i \tag{2.1}$$

- $w_{i,j}^{t}$: is the connection weight between input i and output j on time t.
- x_i: is the i input value of the current training example.

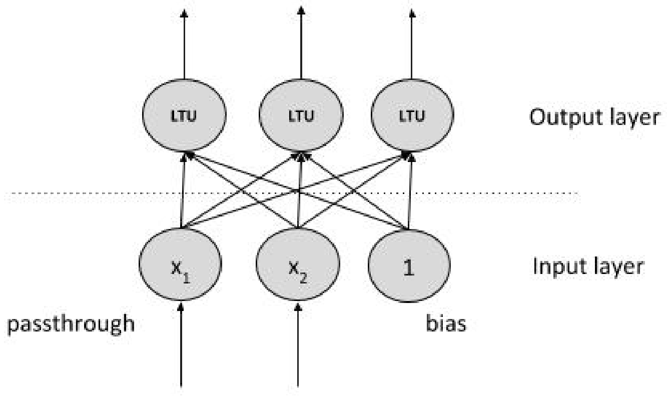


Figure 2.2: Diagram of a Perceptron. A Perceptron is a combination of a number of LTUs, on a single layer. Each input is connected to each LTU, and an additional input, the bias, is added.

- y_j^{gt}: is the expected value for output j.
- y_j^{pred}: is the predicted value for the output j in the current training example.
- α: is the learning rate.

This learning rule is similar to Stochastic Gradient Descent, which is used to train neural networks even at the present day.

Since the output of Perceptrons is linear, it can be proved that they are always capable of solving patterns where the data is linearly separable. However, their representation power is somewhat limited and they are unable of learning complex patterns. For example, they cannot learn a representation where they can correctly classify an XOR function (having two inputs, output is 0 if both inputs are equal and 1 if they are different). This problem is solved by stacking Perceptrons, and the resulting architecture is called Multi-Layer Perceptrons.

2.1.3. Multi-Layer Perceptrons

An MLP is composed of one input layer, one or more layers of LTUs (called hidden layers) and one final output layer. Each output layer also contains one neuron that is a bias, and each neuron is fully connected to the next layer. This means that the output of each neuron is an input to each and every neuron of the following layer. A diagram of an MLP can be seen on Figure 2.3.

One key decision in the definition of the LTU is the choice of the step function. The step function has to be non-linear, so that the output of the network is not

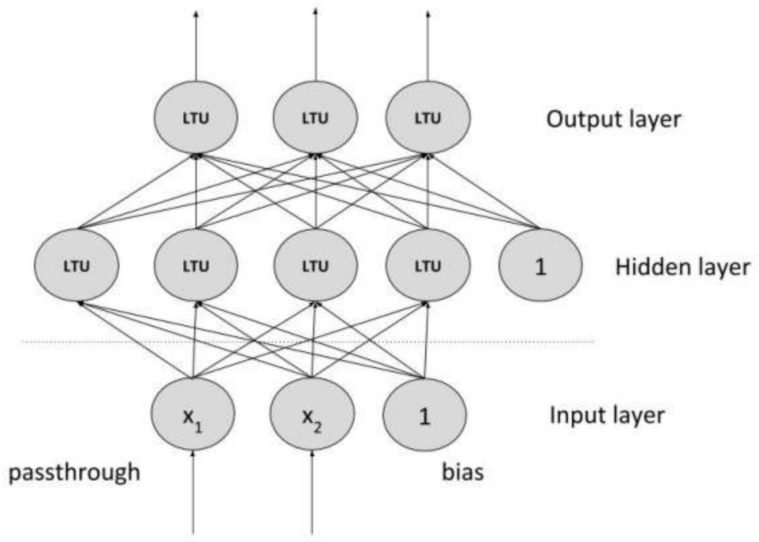


Figure 2.3: Diagram of an MLP. An MLP is a composition of LTUs layers. Inner layers are called hidden layers and the outermost layer is the output layer. Each hidden layer, in addition to the LTUs, contains a neuron that is a bias.

linearly dependent on the input. This non-linearity is what prevents the network from collapse into a unique linear combination and provides the network with more representation power.

Usually, when the network contains two or more hidden layers they are called Deep Neural Networks (DNN). Training them is no different than training any other Machine Learning model: an optimization procedure is defined together with a cost function. The nonlinearity present in these models causes loss functions to become nonconvex. Therefore, these models are usually trained using iterative, gradient-based optimizers that push the cost function to a low value.

2.1.4. Backpropagation

For many years, training these kinds of networks was considered problematic. The introduction of the *Backpropagation* algorithm, in 1986, solved this problem. It enables the application of Gradient Descent. It is done using reverse-mode automatic differentiation (automatically calculating gradients applying the chain rule, from the output layer to the input layer).

In a first stage, we make a prediction on each element of the training set while storing the result of each layer. Weights were previously initialized randomically. There are several ways of initializing the weights, each with a different impact on the network, this will be discussed later. This first stage is called **forward**

propagation or forward pass. A scalar cost $J(\theta)$ is computed as result of this procedure.

The backpropagation algorithm allows the information from the cost function to flow back through the network, to compute the gradient for each weight. Computing an analytical expression for the gradient is straightforward but it can be computationally expensive. Backpropagation allows us to compute the gradients in a cheap and simple way. It is based on the chain rule of calculus, that states that given $y = g(x)$ and $z = f(g(x)) = f(y)$, $\dfrac{dz}{dx} = \dfrac{dz}{dy}\dfrac{dy}{dx}$. Using it, an algebraic expression can be written for each gradient of the cost function with respect to each weight. However, when evaluating these expressions, considerations must be made since a lot of computations are involved.

The gradient of the loss function of the network is calculated. Then, how much each neuron of the last hidden layer contributed to the error of each output neuron's error is measured by calculating the gradient of the outputs with respect to the weights in the last hidden layer. This procedure is iteratively followed on each hidden layer, until the input layer is reached. In this way, the error gradient is propagated backward and calculated in each weight and bias. This is known as **backward pass**. By performing it iteratively in this way unnecessary computations in the chain rule application are avoided.

Calculating the gradient for the loss with respect to each weight, enables the application of Gradient Descent. Gradient Descent is an optimization algorithm that finds the local minimum (maximum) of a differentiable function by taking steps towards the negative (positive) direction of the gradient of the function. A particular variation of gradient descent is usually used when training Neural Networks: Stochastic Gradient Descent. The only difference is that instead of calculating the gradient considering the whole training dataset, it takes an estimate by calculating the gradient from a subset of the data.

2.1.5. Activation functions

One key modification in the MLPs architecture was necessary to be able to apply Backpropagation algorithm. Activation function *step* had to be replaced. The reason for this is that Gradient Descent cannot be applied on flat segments, since gradient is 0 and it cannot "move" through the surface. This function only had flat segments. Currently several activation functions are common:

- Logistic function: $\sigma(z) = \frac{1}{1+exp(z)}$ - Original substitution, but nowadays is rarely used.
- Hyperbolic tangent function: $tanh(z) = 2\sigma(2z) - 1$ - Similar to logistic function (S-shaped output), is continuous and differentiable, but is 0-centered, having its output values between -1 and 1. This makes the output relatively normalized, which sometimes helps speed up convergence.
- Rectified Linear Unit (ReLU): $ReLU(z) = max(0, z)$ - It is continuous and differentiable except on 0. In practice is usually used since it works really

well and it is cheap to compute.

2.1.6. Hyperparameters

Neural networks are quite flexible. This is both an advantage and a disadvantage. There are lots of hyperparameters that need to be adjusted, which can drastically change the output of a network. In the context of neural networks, we call hyperparameters to those set when building the network, and determine its architecture (in contrast to network's parameters that are adjusted during training, like weights and biases). Some of them are:

- Number of hidden layers
- Number of neurons per hidden layer
- Activation functions

Number of hidden layers. The *universal approximation theorem* proves that neural networks with only one hidden layer are universal approximators. This means, they can approximate arbitrarily well any real-value continuous function. Nevertheless, the theorem does not state steps for its construction, it just proves that it is possible.

Neural networks with several layers have better efficiency in the usage of parameters: a network with more hidden layers can exponentially use fewer neurons than a network with only one hidden layer. This makes it both faster to converge and easier to train. In addition to it, it improves generalization on the dataset.

On the other hand, too many hidden layers can make it harder to train. Finding the appropriate number of hidden layers is not currently well defined and it usually depends on the experience of the practitioner (and his luck).

Number of neurons per hidden layer. The number of neurons in hidden layers is tightly connected with the input and output of the network. If they are too many, the network can essentially memorize every input and expected output, failing to generalize. A common technique is to increase the number of neurons until the training dataset is overfitted (predicting this set with 100 % accuracy, without considering the validation dataset), and then apply different regularization techniques, like dropout or batch normalization to make the network generalize.

Like selecting the number of hidden layers, a procedure to find the best value does not exist, and this number has to be selected by a combination of experience and trials and errors.

Activation functions. Although the most used activation function is ReLU, there are several alternatives. While some of them are completely different, others are just variations over ReLU. One of the advantages of this function is that it is extremely cheap to compute, and in practice it works extremely well. This is partly due to the fact that it does not saturate its output on large input values (unlike tanh or sigmoid). We will see different activation functions in one of the following Sections.

2.1.7. Deep neural networks

If we are tackling a complex problem, it may be necessary to increase the complexity of the architecture of the network (e.g., increasing the number of hidden layers or the number of neurons in them) to solve it correctly. However, this increase can lead to several different problems:

- Vanishing and/or exploding gradients: Backpropagation algorithm to train the network propagates the error from the output layer to the input layer. If we are dealing with a large network, gradients tend to become small (vanishing) or quite large (exploding) on each subsequent pass of the algorithm to the next layer.
- Training time: If the network is large and it contains millions of parameters, training time can become really slow and a bottleneck in the development of a solution to the problem.
- Training set memorization: a large number of parameters risks overfitting the training set and not generalize properly.

Vanishing and exploding gradients. The backpropagation algorithm works by calculating the gradient of the cost function with respect to the output layer, and then moving towards the input layer by propagating this error gradient in order to calculate the error gradient corresponding to each parameter. In each step of gradient descent, parameters are updated.

Gradients usually become smaller as the algorithm progresses towards the input layer. The effect of this is that weights on the lower layers (those closer to the input layer) are virtually unchanged, and it may take a lot of iterations to converge to a solution (or it might even not converge at all). The opposite can happen as well, where gradients increase towards the lower layers, weight updates are huge and the algorithm diverges. This is more common in Recurrent Neural Networks. We will see now some techniques to alleviate these problems.

Initializations. On a 2010 paper, Glorot and Bengio [27] analyzed and concluded that the standard way of initializing the weights together with the activation function choice of logistic function was one of the main reasons of the difficulty in training these networks. Weights where randomly initialized by taking samples of a normal distribution with mean 0 and standard deviation 1. In that paper, they showed that in that case the variance of the output is much greater than the variance of the input. It increases on each layer until the activation function saturates on the top layers.

When the input becomes large (both negative and positive), the logistic function saturates at either 1 or 0, and its derivative is extremely close to 0. Therefore, when backpropagation is applied, gradient is extremely close to 0 and it cannot be propagated properly through the network. This problem is worse on each layer since the gradient keeps getting diluted on each layer.

In order to solve this problem, an initialization was proposed to allow the signal to flow in both directions. Since the objective is for the signal not to die out but also not to explode, the authors argue that the variance of the outputs must be equal to the variance of its inputs and also the gradients must have equal variance before and after flowing through each layer. Then, weights are initializated randomly sampling from:

- A normal distribution with mean 0 and standard deviation $\sigma = \sqrt{\frac{2}{n_{inputs}+n_{outputs}}}$
- A uniform distribution between -r and r with $r = \sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$

While it does not guarantee that both conditions are met (and this is only possible when $n_{inputs} = n_{outputs}$) it is a good compromise that works well in practice. These distributions depend on the choice of the activation function. The former are known as *Xavier initialization.* Another well-known initialization technique is the one used for the ReLU activation function, known as *He initialization* [33]:

- Normal distribution with mean 0 and standard deviation $\sigma = \sqrt{2}\sqrt{\frac{2}{n_{inputs}+n_{outputs}}}$
- Uniform distribution between -r and r with $r = \sqrt{2}\sqrt{\frac{6}{n_{inputs}+n_{outputs}}}$

2.1.8. Activations

The same 2010 paper [27] shown that part of the vanishing/exploding gradients problem was due to the choice of the activation function. After that paper, logistic activation function fell slowly out of fashion and was replaced by ReLU, mostly because it does not saturate on positive values and it is fast to compute.

$$ReLU(z) = max(0, z) \tag{2.2}$$

While they behave better than the logistic, they suffer from another set of problems. In particular, they can *die* during training and output only 0s. This happens after an update where the weighted sum of the neuron's input is negative. In this case, it is unlikely that it will step out of this state. To solve this, LeakyReLUs where proposed:

$$LeakyReLU(z) = max(\alpha z, z) \tag{2.3}$$

Other similar variants of ReLUs where proposed, such as randomized ReLUs, parametric ReLUs, and perhaps the most interesting one, exponential linear units (ELUs), that seem to outperform the other variants. However, it is more expensive to compute.

2.1.9. Optimizers

Having computed the analytic gradient with backpropagation, this result is used to update the parameters. While the basic approach is to apply gradient descent, there are several other variations. The basic update is done by: $x = x - \alpha \cdot dx$, where x is a vector of parameters, α is the learning rate (the length of the step taken) and dx is the gradient of the loss function with respect to the parameters x.

Momentum is an alternative to this simple approach, that consist of adding a term that takes into the account the movement that the algorithm has had in previous iterations. If it is moving in some direction, it builds up a certain velocity. The update rule is then: $x = x - v$ where $v = \mu * v - \alpha \cdot dx$. μ serves as a coefficient of friction, usually taking values between 0.5 and 0.99, to slow the velocity. These changes improve convergence times for the algorithm.

Nesterov momentum [6] is an improvement on the previous approach. Basically, the calculated momentum is going to modify the step taken in a way that is known before calculating the gradient. Therefore, instead of calculating the gradient in the original position, it gets calculated in the modified position (also called look-ahead position), getting in this way a more precise gradient step. Then, the update has the form: $x = x - v$ with $v = \mu \cdot v - \alpha \cdot dx^{ahead}$ where $x^{ahead} = x + \mu \cdot v$.

These methods use a unique learning rate through the whole process and in the same way for all parameters. Since the learning rate is a hyperparameter that is expensive to tune, methods have been devised to automatically adapt the learning rate as part of the learning process.

In **Adagrad** [22], a cache the size of the gradient is kept. It is used to keep track of the sum of the squared gradients: $C = C + dx^2$. This cache is then used to normalize the update step per parameter: $x = x - \frac{\alpha \cdot dx}{\sqrt{C} + \epsilon}$. In this way, weights that receive high gradients get their learning rates reduced and viceversa. The ϵ term is added to avoid divisions by zero.

RMSProp is a modification of Adagrad, because the latter is sometimes too aggressive in decreasing the learning rate. The modification is done on the cache: $C = \gamma \cdot C + (1 - \gamma) \cdot dx^2$. In this way, the learning rate of each weight depends on the gradients it receives, but since the cache is leaky (due to the first term), learning rate do not get monotonically smaller.

Finally, **Adam** [44] is a combination of RMSProp and Momentum. Parameter updates look like in RMSProp or Adagrad: $x = x - \frac{\alpha \cdot m}{\sqrt{v} + \epsilon}$. m is a smoothed version of the gradient: $m = \beta_1 \cdot m + (1 - \beta_1) \cdot dx$ and v acts as the cache, storing the momentum per weight: $v = \beta_2 \cdot v + (1 - \beta_2) \cdot dx^2$

In practice, Adam is heavily used and recommended, and it is usually the one that yields best results. However, running tests at the same time with SGD together with Nesterov is also common.

2.1.10. Regularization

Regularization is any modification made to reduce the generalization error (test error) but not the training error. It might even be detrimental to the train error. It tries to improve the generalization of the algorithm, this is, the performance on unseen cases. It is a way of dealing with the problem of overfitting the training set. There are many regularization strategies. To name some examples: some of them put constraints on the values that the parameters can take, others add terms to the objective function, and others modify the training algorithm. This list is by no means exhaustive. We will discuss two techniques that are heavily used in deep learning: Dropout and Batch Normalization.

Dropout. Dropout [82] is a computationally cheap strategy to regularize a wide variety of models. The key idea in dropout is to randomly drop non-output units and their connections with some probability, usually between 20 % and 50 %. This prevents the network from relaying too much on a particular neuron, an input, and on units to co-adapt too much. This strategy yields great results in terms of regularization, is simple to implement and is fast to execute.

When making predictions, units are not dropped. Therefore, the output of the neurons must be scaled to have the same expected outcome as the one that they had in training time. For example, given a neuron with output x and dropout rate p, the expected output would be $px + (1 - p)0$. At inference time, the output would be x, so to have the same input that it was receiving at training time, x has to be adjusted to px. Basically the activations are adjusted by the dropout rate. In practice, instead of adjusting at inference time, this is considered at training time and outputs are adjusted at that point. This is partly because to simplify the implementation of the prediction, which has the advantage of being faster and transparent to the techniques applied at training time.

Batch Normalization. Vanishing and exploding gradient problems can come back during training. In a 2015 paper [40], Ioffe and Szegedy proposed a technique to address them, called Batch Normalization. The distribution of the layer's input changes during training as the parameters of the previous layer changes (they name it Internal Covariate Shift), and this technique tries to solve that.

Essentially, they add an operation before each activation which consists in zero-centering and normalizing the inputs, and then scaling and shifting the result using two new parameters (one for scaling and the other for shifting). In this way, the model learns the optimal scale and shift of each input. Mean and standard deviation needs to be estimated to center and normalize the inputs, so they are calculated as a moving average, each time a mini-batch is fed to the network. Of course, the downside is of increased complexity in the network and also a runtime-penalty due to the extra computations.

2.2. Convolutional Neural Networks

Several different neural network architectures have been proposed throughout the years, with different results depending on the task at hand. Convolutional neural networks are particularly suited to process data that has a grid-like topology - locality is important. They are particularly suited to process images, given that this kind of data possesses this characteristic: pixels that are next to each other have a much more meaningful relationship than those that are far away. Nevertheless, they are also applied in other types of data (e.g., Time-series).

CNNs are heavily inspired by the visual cortex of living organisms. In particular, works by Hubel and Wiesel in the 50s and 60s [37], [38], understanding the visual cortex of cats and monkeys, are usually cited as inspiration for the design of these networks. They were initially introduced by Fukushima in 1980 [26], with the proposal of the neocognitron. With this network, two different types of layers were introduced: convolutional layers and downsampling layers. These two remain as key building blocks of CNNs even nowadays. First results with this type of networks were presented by LeCun [50], applying them to understanding zip-code handwritten characters, while also learning the kernels with backpropagation and gradient descent. His research continued for several years, until the introduction of LeNet-5 [49] in 1998. This network was used to recognize handwritten numbers in checks, and became foundational in further research in computer vision problems.

2.2.1. Architecture

The neurons of the first convolutional layer are not connected to every pixel in the input image. They have a small receptive field. In turn, those in the second layer are also connected to a small rectangle in the first layer. The result of this architecture is that the lower layers concentrate on low-level features, while higher-level layers assemble them into higher-level features. A representation of this process can be seen on Figure 2.4.

Weights in convolutional layers are 2D, and they are usually square matrices. The operation made is a cross-correlation, where these matrices are multiplied by the input, and then every value of the result is summed up to output just one value. These 2D weights are called filters or convolution kernels, and the result of the cross-correlation operation of each filter with each part of the image (receptive field) is called a feature map. Filters can be stacked together, resulting in several stacked feature maps. Within one feature map, all neurons share the same parameters (weight and biases), but different feature maps have different parameters.

Having all neurons in a feature map share the same parameters reduces the number of parameters in the model drastically, but most importantly, it means that once the CNN has learned to recognize a pattern in one location, it can recognize it in any other location. CNNs are translation-invariant.

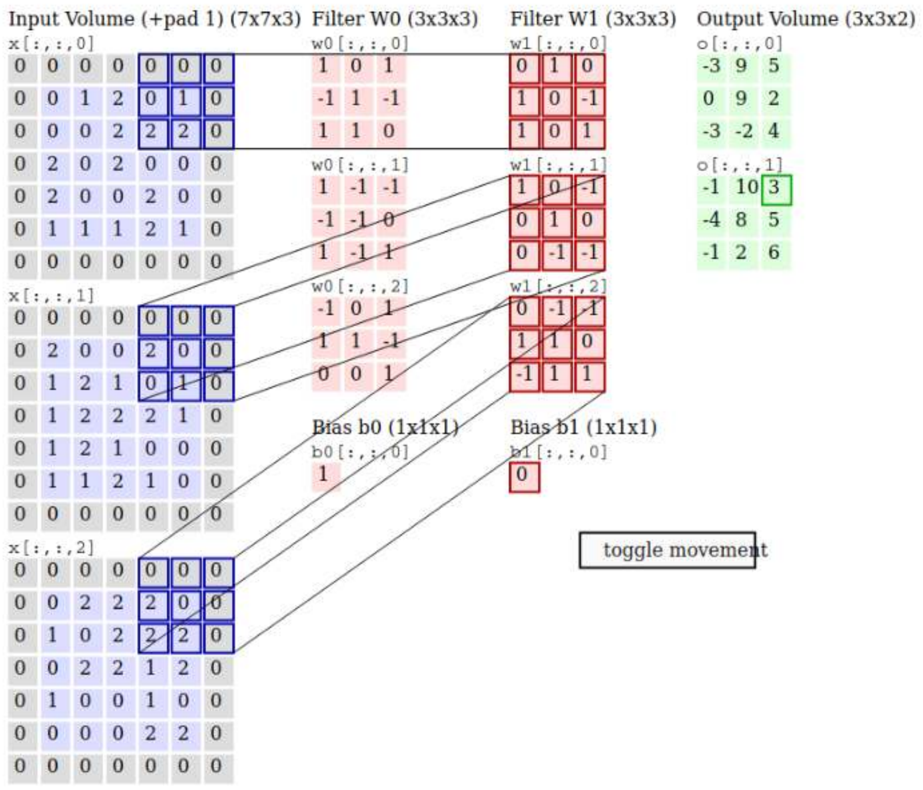


Figure 2.4: Representation of the process in a convolutional neural network. Each filter ($W0$ and $W1$) in turns, are slided over the input volume. In the step seen in the image, the filter $W1$ is multiplied by the uppermost right part of the input volume. The sum of the result of the multiplication (which is a cross-correlation), fills its corresponding value in the output volume. Image taken from Stanford course cs231n [95].

Pooling Layers. The goal of pooling layers is to subsample the input image, in order to reduce the computational load, memory usage and number of parameters. It also helps the CNN tolerate a bit of image shift. Pooling layers, like convolutional layers, are connected to a small rectangular receptive field. However, they have no weights. All that it is done is to aggregate the inputs using a function like max or mean. Pooling layers typically work on each channel independently. The two most common pooling layers are MaxPooling and AvgPooling which use max and mean as the aggregation function, respectively.

2.2.2. CNN architectures

Several architectures were proposed over the years. Most of them build on the previous proposals, introducing new changes to get better results on different tasks. We will discuss a few examples that had some major significance, either because they were introducing a new important idea or because they were the best performing alternative when proposed.

LeNet5. Presented in 1998 [49], it was one of the first CNN that achieved great results. It is considered to be a breaking point in the field, having triggered the interest of many researchers in the field, and the inspiration of further advancement that would come in later years. It is a simple architecture, containing two convolutional layers and three fully connected layers, with a total of 60K parameters.

AlexNet. Another important breakthrough in the field was the proposal of AlexNet [45] in 2012. It was presented in the ImageNet Large Scale Visual Recognition Competition, and it won by a large margin over its contenders, which started a trend of seeing increased application of CNNs in submissions to this type of challenges. It contains 8 layers, 5 of them being convolutional layers while the remaining 3 are fully connected. Being significantly larger than LeNet, it contains 60M parameters. Having this number of parameters, its training was computationally expensive, and it was made possible due GPUs usage. This was a trend that would continue to increase in the following years. Another interesting innovation was that they changed the activation function, using ReLUs, which was something not common at the time.

VGGNet. VGGNets [80] are another iteration on the same trend, making networks deeper, while maintaining pretty much the same architecture. The main contribution in this paper is moving towards deeper architectures, which proved to yield better results. They presented different variants of the networks: VGG-13, 16, and 19, with 10, 13, and 16 convolutional layers respectively, and 3 fully connected layers. VGG16, which was greatly used in other problems (both its architecture and its weights with transfer learning), is comprised of 138M parameters.

ResNet. Introduced in 2016 [34], ResNets addressed a problem that CNNs where suffering at the time: Increasing the depth of the networks improved the accuracy up to a point, where it would start to saturate and then it would degrade rapidly. This architecture introduced (or actually popularized) the usage of skip connections (also called residual connections). These skip connections link the input of a layer with the output of a layer two or three levels before. These connections simplify the network and reduce the impact of the vanishing gradients problem. Combined with these, this network was also an early adopter of Batch Normalization, benefiting from its advantages.

Inception & Xception. Around at the same time ResNet was proposed, another family of architectures were proposed between 2014 and 2016. These are Inception-v1 and Inception-v3, in 2014 [88], and 2015 [89], respectively, and Xception in 2017 [11]. The main idea behind both inception networks, was to improve the usage of computational resources inside the network. This is done by the usage of 1x1 convolutional layers, to reduce computational bottlenecks, and using parallel towers of convolutions with different filter sizes, to cluster different features. v3 performs some variations to further improve the usage of computational resources by factorizing some convolutions. Xception takes the inception idea to the extreme replacing the inception modules with depthwise separable convolutions. 1x1 convolutions are applied to each channel, thus capturing cross-channel correlations. Then 3x3 or 5x5 convolutions are applied, capturing spatial correlations within each channel.

Others. CNN architectures are an active area of research, and lots of other architectures were presented after the ones mentioned above: Inception-ResNet-v2 [87], ResNeXt-50 [99], Efficient-Nets [92], etc. In this Chapter we reviewed some of the key contributions that where made over the years, and that have had impact in different problems in Computer Vision. In the next Chapter, we will focus in an specific area of study in Computer Vision: Face Recognition.

Chapter 3

Face recognition

Face recognition is a computer vision problem, in which we try to match the identity of a person with a picture of her face. Two subproblems can be considered:

- Face verification: Given an image of a face, it gets compared to a ground truth image of a person's face, and the challenge is to state if it is the same person or not to verify its identity. It can be used, for example, as a biometric safety measure to unlock a device such as a smartphone.

- Face identification: Given an image of a face, and a database of images of faces with associated identities and other information, the top-N most similar ones are retrieved, to try to identify the person in the original image. It can be used with missing people databases, with law enforcement or with personalization of systems. The usage of these systems and the collection of facial images, has been under heavy scrutiny due to privacy issues. We keep these important ethical and practical considerations very present during all our research.

In this Chapter, we will study the problem of face recognition and techniques used in this area. Then, we will review Active Stereo Geometry fundamentals, which will be necessary for Chapter 4.

3.1. Face Recognition

The process of identifying or verifying a person's identity only using an image of their face is a complex problem. How an image of a face is captured may impact greatly in the accuracy of the system. For example, resolution, lightning, head rotation or tilt affect the performance and the ability of the system to match two faces (and therefore, two identities). Aging, occlusions, styles (e.g., facial hair-style) are also variables to be considered. While results considering only some of the problems presented above, achieve an extremely good accuracy (e.g., recognition made from ID photos), other variables like aging, occlusions and lighting have proven to be more challenging. They are still under active research.

The general framework used for face recognition consist of three main steps: detect the face, extract representational features and finally, compare those features.

3.1.1. Face detection

It can be considered a specific case of object detection, in which the object to detect is a face. The objective is to detect the presence of a face in an image, and if detected, determine the location inside that picture. Usually, that location is given by defining a bounding box that encloses the face with the pixel-location of the four corners of the box, or stating the pixel-location of facial landmarks, such as the corners of the eyes, nostrils, points on the chin, etc. It is a problem in itself since it has applications other than face recognition.

Like face recognition, detection is a problem that has been studied for several years now, with proposed techniques like Eigenfaces in 1987 [81] and Viola-Jones in 2001 [96]. In latest years, new techniques based on CNN have been proposed, improving substantially the state of the art. In 2016 MTCNN [101] was proposed, in 2017 was Faceness-Net [106], and in 2019 RetinaFace [15] which is the best method available today.

After faces are detected, and landmarks are identified, images are transformed, trying to obtain a canonical alignment, in order to remove differences due to translation, scale and rotation. For most unconstrained problems, faces are detected "in the wild" (no predetermined position of the face and camera). Therefore, this step in the recognition pipeline has a significant importance to enable the extraction of representative features of the face. To be representative, they must be invariant to affine transformations. If the modifications needed for the problem at hand are small (small rotations, translations), using simple geometrical transformations may be enough. However, when dealing with datasets that contain faces in any position, more complex techniques must be applied.

We will present RetinaFace in the following paragraph, as an example of one of these techniques, to see the challenges and techniques that this task involves.

RetinaFace. RetinaFace combines a lot of techniques and improvements made over the years in other publications, together with some original proposals.

Basically it works by applying feature pyramid levels on the different stages of a ResNet. It gets feature levels at different scales, using top-down and lateral connections. On each level, a set of anchors are defined where to search for faces. 75 % of the anchors come from the lowest level, which correspond to tiny faces. Then, that information is combined in context modules, one for each level. What these context modules do is to enhance the receptive field and the rigid modelling power. A representation of this architecture can be seen on Figure 3.1.

After processing this information, if the anchor corresponds to a negative example (meaning it does not correspond to a face), only a simple classification loss is applied. However, if the anchor corresponds to a positive sample, a multi-task loss is applied. The application of this loss is what has improved the result of the

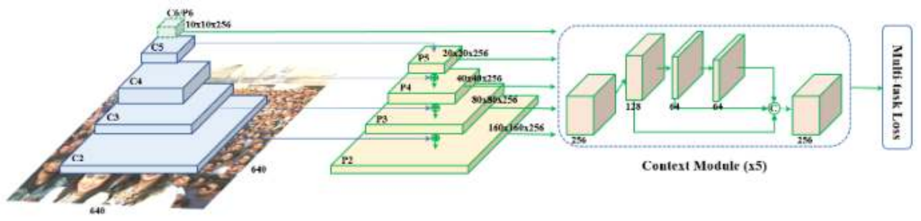


Figure 3.1: RetinaFace architecture. Applies feature pyramid levels on different stages of a ResNet. It uses top-down and lateral connections to get to P2 to P6 feature levels. P2 to P6 are fed to context modules. Image taken from [15].

Average Precision in face detection.

The loss is composed in the following way:

$$L = L_{cls}(p_i, p_i^*) + \lambda_1 p_i^* L_{box}(t_i, t_i^*) + \lambda_2 p_i^* L_{pts}(l_i, l_i^*) + \lambda_3 p_i^* L_{pixel} \tag{3.1}$$

L_{cls} corresponds to the classification loss, where pi is the prediction of the anchor being a face or not, and p_i^* is 0 for a negative anchor and 1 for a positive anchor. L_{box} corresponds to the regression loss for the bounding box around the face, which is calculated with an smoothed-L1 loss. L_{pts} is the loss of the predicted facial loss. As part of what was proposed in this paper, they manually annotated 5 facial landmark points: eyes centers, nose and the sides of the mouth. This extra signal proved to improve the performance of the face detection. This loss corresponds to the difference between the predicted points and ground truth points.

Finally, as part of the original proposals, they added a branch to predict the shape of the face in 3D, in parallel to the other predictions (classification, bounding box, facial landmark points). This loss, L_{pixel}, corresponds to the difference between the prediction of the 3D shape and the prediction of the 3D shape done by a different model (mesh decoder), which is taken as ground truth. In an ablation study done also in this paper, they showed that the presence of this branch also improves the performance.

As a result, the proposed method improves the state of the art performance in WIDER FACE [100] dataset and also, when used together with ArcFace, improves the state of the art performance in IJB-C [62] dataset for face verification.

3.1.2. Recognition architectures

Different recognition techniques were proposed throught the years. We will analyze some examples presented in the last decade. Based on CNN architectures and trained on large datasets, these networks have achieved astonishing results, enabling its usage in different applications, even for the general population.

DeepFace. Deep face [90] was introduced by Facebook in 2014. On this paper, they state that the modern face recognition pipeline has 4 stage: Detect $\rightarrow$ Align $\rightarrow$ Represent $\rightarrow$ Classify. They used a Deep Neural Network with 120 million parameters for the steps for Align and Represent. They improved by 27 % the accuracy in LFW [36] (unconstrained Face Recognition).

It was one of the first to use DL for this task, specially considering such a heavy improvement on the accuracy of the current face recognition benchmark dataset.

Their contributions can be summarized as follow:

1. Development of a DNN that is trained on a very large labeled dataset and obtains a representation that generalizes well to other datasets.

2. An effective face alignment system based on 3D modelling of the faces.

3. Advancements in state of the art in LFW reaching near human-performance levels and YTF [98] decreasing error rate in more than 50 %.

They developed a novel 3D-alignment technique to deal with out-of-plane rotations that affect the recognition performance.

To perform the alignment, they start by detecting 6 fiducial points on the detection crop, that correspond to the eyes (2), nose (1), sides and center of the mouth (3). This detection is done with a simple Support Vector Regressor (SVR). With these points, a 2D similarity transformation (scale, rotate and translate) is calculated, taking these points to 6 anchor locations. Iterations on this process are made with the new transformed image until no significant change is detected.

Then, on the resulting image, 67 fiducial points are detected, again with an SVR. A generic 3D shape model is used (which is an average of samples from a database with 3D scans of faces) in which 67 fiducial points were manually placed. Using the correspondence between the two sets of points in 2D and 3D, an affine 3D-to-2D camera (P) is fitted.

Finally, with P calculated, a step that they call *frontalization* is made. A relaxation of the target 3D points is made by adding the residuals of the previous step to each fiducial point. This is because P is only an approximation and in applying the transformation, identity-bearing factors would be distorted by the warping made. Then, considering the Delauny triangulization from the 67 fiducial points, a piece-wise affine transformation from 2D to 3D is made. The different intermediate steps of the process can be seen on Figure 3.2.

A DNN is trained on a multi-class face recognition task. The DNN takes as input the 3D-aligned RGB image. Its architecture, which can be seen in Figure 3.3, consists of 7 layers. C1 and C3 are convolutional layers, with 32 11 by 11 filters and 16 9 by 9 filters respectibly. M2 is a max pooling layer to be more robust to local translations. L4, L5 and L6 are locally connected layers: they apply a filter bank but every location on the feature map learns different filters. Finally, F7 and F8 are fully connected layers, with F7 being the representation vector for the face, while F8 is connected to a softmax layer [29], used only for the purposes of training. Dropout is applied on F7. As a last step, the resulting vector from F7 is normalized to zero to one range, and then an L2-normalization is applied.

In order to generalize to different datasets, the verification metric for two faces that they use is just the inner product of two normalized feature vectors. However,

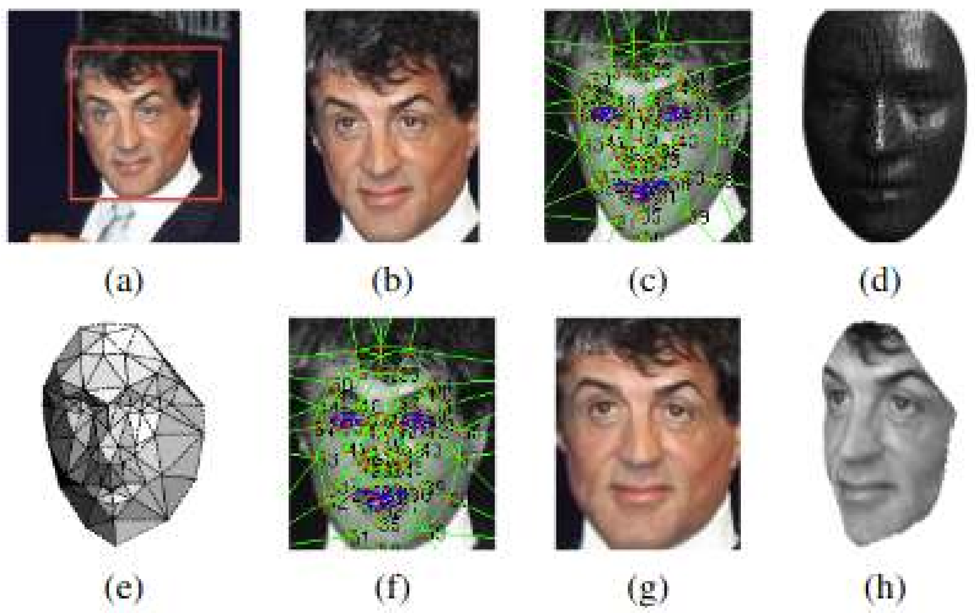


Figure 3.2: Alignment pipeline. (a) The detected face, with 6 initial fiducial points. (b) The induced 2D-aligned crop. (c) 67 fiducial points on the 2D-aligned crop with their corresponding Delaunay triangulation, triangles are added on the contour to avoid discontinuities. (d) The reference 3D shape transformed to the 2D-aligned crop image-plane. (e) Triangle visibility w.r.t. to the fitted 3D-2D camera; darker triangles are less visible. (f) The 67 fiducial points induced by the 3D model that are used to direct the piece-wise affine warping. (g) The final frontalized crop. (h) A new view generated by the 3D model. Image and caption taken from [90].

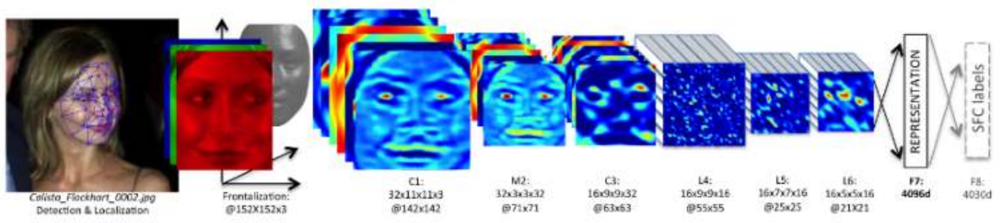


Figure 3.3: DeepFace architecture. The CNN architecture is composed of 2 convolutional layers (C1 and C3), one max pool layer between them (M2), and then 3 locally connected layers (L4, L5, and L6). Finally, F7 and F8 are fully connected layers. F7 is used as the representation vector. The input image is frontalized as explained above. Figure taken from [90].

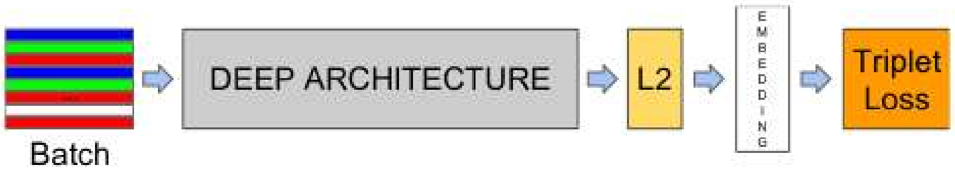


Figure 3.4: FaceNet structure. A batch of images is fed to a CNN architecture. Its output is L2 normalized, getting the output embedding. During training, the embedding is used to calculate the triplet loss. Image taken from [77].

in addition to that metric, they made tests with a weighted X^2 distance and with siamese networks.

This DNN is trained on a private dataset called Social Face Classification presented on this paper, built from Facebook images. It contains 4.4M images of 4K identities. They performed several tests, achieving really good results on LFW and YTF. The best results were achieved by an ensemble of DNNs, combining a DNNs with different inputs: four with the 3D aligned image trained with different random seeds, one with the 2D aligned image, one with the gray-level image plus image gradient magnitude and orientation and one siamese network.

It was an important paper that accelerated the usage of deep learning in the face recognition task.

FaceNet. FaceNet [77] was introduced by researchers at Google by mid 2015 (at that year's CVPR). It's main contribution is that the system learns directly a mapping from faces images to an Euclidean space, where distances on that space correspond to a measure of the similarity of faces, while requiring minimal alignment. In this way, to perform tasks of classification or verification, these embeddings can be used and then only a threshold needs to be defined (that separates different classes, different identities). Previous methods (like DeepFace), needed to add a classifier such as SVM on top, or perform some post-processing like PCA.

As a result of these improvements, they get a better representational efficiency, since they use feature vectors of 128 dimensions, while improving the state of the art in LFW and YTF.

While other methods used bottleneck layers in the network as a representation, FaceNet objective is to learn this representation. Previous methods were indirect and inefficient in the sense that the generalization to other faces is not always ensured and that those layers were usually larger (around 1000 dimensions). To learn this 128D embedding as the output of FaceNet, it is trained with a triplet-loss function. The general structure of the system is as shown in Figure 3.4.

The triplet loss essentially involves taking triplets of images, where one is the anchor, and the other two are one positive example (same identity) and one negative example (different identity). The idea is for the squared distance of the anchor to the positive example to be smaller than the squared distance from the anchor to the negative example plus a margin. Therefore, the loss minimized is the following: $L = \sum_i^N [||f(x_i^a - x_i^p)||_2^2 - ||f(x_i^a - x_i^n)||_2^2 + \alpha]$. An idea of the process

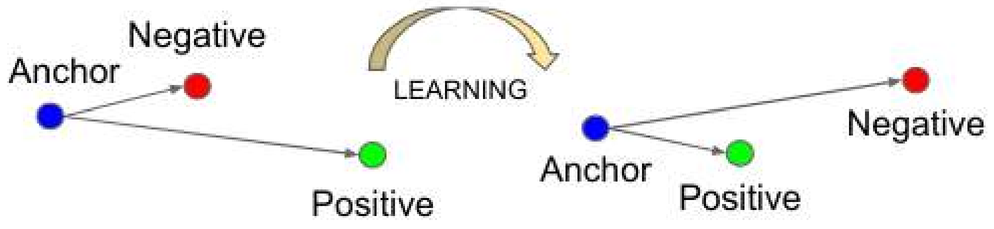


Figure 3.5: Triplet Loss process. Through training the network, the embeddings corresponding to the anchor and the negative example are separated while the embeddings corresponding to the anchor and the positive example are pulled together. The process of selecting triplets to calculate the loss is extremely important to achieve good results. Image taken from [77].

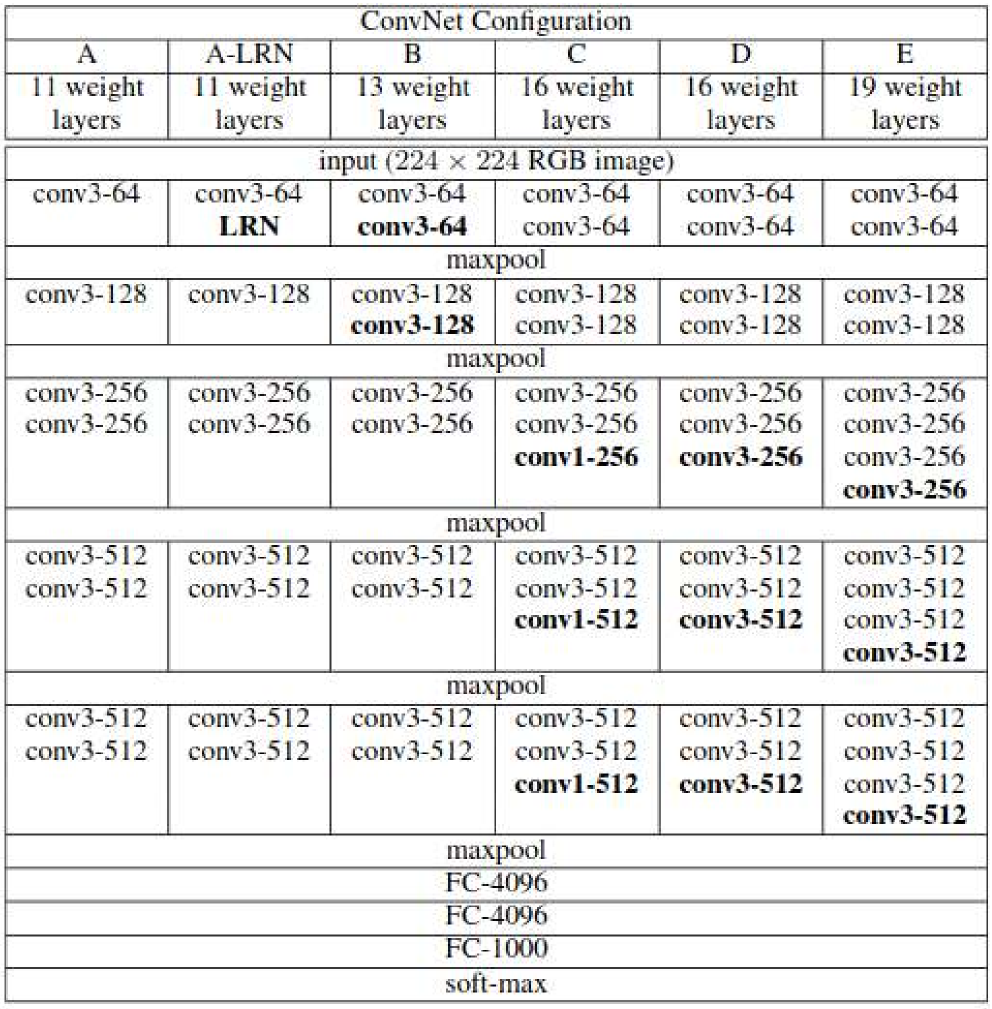
can be seen in image 3.5.

Some considerations must be taken into account when selecting the triplets, in order to get a fast convergence. Selecting all the possible triplets is impractical since they do not contribute to the training but still take time. The goal is to select hard examples. They provide two alternatives: generating the triplets offline or online on each mini-batch. They focus on the online method. On each mini batch, they use all the combinations of positives, while only selecting hard negatives: those with the smallest distance to the anchor. They select examples with the smallest distance to the anchor while also fulfilling that $||f(x_i^a - x_i^p)||_2^2 < ||f(x_i^a - x_i^n)||_2^2$, that is, that they are inside the margin. This is because only selecting the hardest examples can lead to bad local minima and a collapsed model. Due to the use of this loss, they have to train with batch sizes close to 1000 examples, ensuring having a minimum number of positive examples in each batch.

Regarding the architecture of the CNN, they performed several tests. They used one architecture based in Zeiler&Fergus [105], that has 140M parameters. They also used a set of architectures based on Inception [88], ones that have 7M parameters and others that have around the same number of parameters but fewer FLOPS per image, focused on running on mobile phones. Every network is trained on 100M-200M images, of 8M identities.

Best results were achieved with one of the Inception architectures, improving the state of the art in LFW and YTF. They also provided some analysis regarding the number of parameters, FLOPS, amount of training data, image quality and embedding dimensionality.

VGGFace. VGGFace [69] paper was introduced by the end of 2015, by the Visual Geometry Group in Oxford. This paper can be considered in response to DeepFace and FaceNet papers (together with papers about DeepId's [83], [84], [85], [86]), because they rightfully state that the recent advancements in Face Recognition are in part due to the usage of extremely large datasets, which are impossible to get for research groups. Datasets used by Facebook and Google had 4.4M and 200M images respectively, while the largest public dataset at that moment had 200K images.



ConvNet 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 (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 3.6: VGG architectures. The different configurations proposed in [80] are shown. While they have similar structures, complexity increases to the right, adding more convolutional layers. In VGGFace, A, B and D are used. Image taken from [80].

In this paper they present a procedure to gather a large dataset (2.6M images of 2.6K identities) which they made publicly available, and also they presented as simple architecture that was on par with the state of the art on LFW while improving it in YTF. This was also in contrast with the latest DeepId papers, that proposed ensembles of 200 CNN and were the reigning state of the art in LFW at that time. We will omit reviewing the construction of the dataset.

The architecture that they used in their experiments is based on the VGG network [80], which was the winner of some tasks in the 2014 ImageNet challenge. Configurations A, B and D for VGG are used. They differ on the amount of weight layers they have, with 11, 13 and 16 layers respectively. They can be seen on Figure 3.6.

Initially they train using a classifier on top with a softmax score. They use the dataset that they created, that contains 2,622 identities. There is no overlap between the identities in that dataset and LFW or YTF. Each image in the dataset is rescaled to 256x256, and during training random 224x224 crops are taken (from each corner and the center) and also flipped horizontally with a 50 % probability. 2D face alignment is performed.

The last fully connected layer (FC-4096) can be used as a representational feature to perform face verification comparing euclidean distances. However, the results can be greatly improved by fine-tunning them using a triplet-loss (as in FaceNet). This two-stage training is done because they found that bootstrapping the CNN with a classifier on top made the training significantly easier and faster.

To learn the optimal projection with a triplet-loss, they L2-normalize the output of the previous net and add a FC-layer of dimension 1024. The triplets are selected by considering all the (anchor, positive) pairs, and extending them with a negative example, sampled randomly from those that violate the triplet-loss margin. The sample is done randomly instead of selecting the maximally violating example in order not to swamp the network with bad examples. To train this layer, the rest of the network is frozen. Best results were achieved with configuration D (VGG-16) and performing embedding learning, getting to values on par with the state of the art in LFW, and actually improving it in YTF. At the same time, the network architecture is much simpler than those that were being used at that time. VGG-16 would end up being used a lot in different tasks because it offers a good compromise between performance and complexity.

ArcFace. ArcFace [14] is one of the newest papers about Face Recognition. It was presented in 2019 by Deng, Guo et al. The paper focuses on proposing a new loss function to maximize the discriminative power of the learned deep features by the DCNNs. In this regard, they propose an Additive Angular Margin Loss, which they call ArcFace. It builds up on previous losses such as SphereFace [57] and CosFace [97]. These losses, that share some similarities, are an improvement over softmax loss and triplet loss. However, in this paper, they present ArcFace as the best of the group.

ArcFace has the advantage of having a clear geometrical interpretation, as it corresponds to the geodesic distance in a hypersphere, while at the same time improved the metrics of the state of the art in several benchmark datasets. On top of this, the computational overhead is minimal.

ArcFace works in the following way. The softmax loss is defined as:

$$L_1 = -\frac{1}{N}\sum_{i=1}^{N} log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}} \tag{3.2}$$

- x_i is the deep feature of the i-th sample.
- y_i is the correct class for sample i.
- W_j is the j-th column of the weight matrix W. If $j = y_i$, the column is the one corresponding to the correct class.

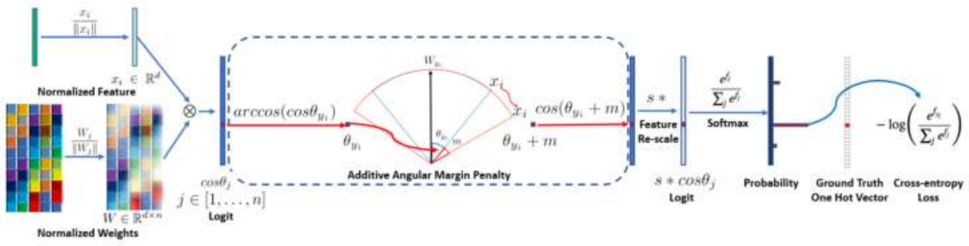


Figure 3.7: DCNN architecture with ArcFace loss. Based on the feature x_i and weight W normalization, the $\cos\theta_j$ (logit) is calculated for each class as $W_j^T x_i$. $\arccos\ \theta_{y_i}$ is calculated to get the angle between the feature x_i and the ground truth weight W_{y_i}. In fact, W_j provides a kind of centre for each class. Then, an angular margin penalty m is added on the target (ground truth) angle θ_{y_i}. After that, $\cos(\theta_{y_i} + m)$ is calculated and all logits are multiplied by the feature scales s. The logits then go through the softmax function and contribute to the cross entropy loss. Image taken from [14].

- $b_{y_i}$is the bias for class y and sample i.
- N is the batch size
- n is the number of classes

Bias is set to 0 and the logit is transformed as $W_j^T x_i = ||W_j^T||||x_i||cos\theta_j$ where θ_j is the angle between the feature vector and the weight vector. W is L2 normalized and the embedding feature x is also L2 normalized but also re-scaled to s. In this way, by normalizing both vectors, the prediction depends only on the angle between them. The embeddings are distributed on a hypersphere of radius s.

Finally, an angular margin m is added to maximize intra-class compactness and inter-class separability. This margin penalty corresponds to the geodesic distance in the hypersphere.

The resulting ArcFace loss is the following:

$$L_3 = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{s\ cos(\theta_{y_i}+m)}}{e^{s\ cos(\theta_{y_i}+m)} + \sum_{j=1,j\neq y_i}^{n} e^{s\ cos\theta_j}} \tag{3.3}$$

.

In the paper, ArcFace is compared to SphereFace and CosFace. It is interesting to understand their difference. They all propose angular margins, but in different ways: SphereFace proposes a multiplicative angular margin (m_1) while CosFace proposes an additive cosine margin (m_3). A combination of all the margins would result in the following loss:

$$L_4 = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{s\ (cos(m_1\theta_{y_i}+m_2)-m_3)}}{e^{s\ (cos(m_1\theta_{y_i}+m_2)-m_3)} + \sum_{j=1,j\neq y_i}^{n} e^{s\ (cos(m_1\theta_j+m_2)-m_3)}} \tag{3.4}$$

.

Other losses proposed are Intra-Loss, Inter-Loss and Triplet-Loss, where terms are added to L2 (L2 is equal to L3 without the margin added, it is omitted here), to improve the intra-class compactness, maximize the intra-class discrepancy and maximize the angle between the triplet samples, respectively.

They performed an substantial amount of experimentation with different configurations. They used two CNN configurations: ResNet 50 and ResNet 100, appending after the last convolutional layer a BN-Dropout-FC-BN structure to get the final 512-D embbedding. They trained on CASIA on some scenarios and MS1MV2 on others. We refer to the paper to analyze the different results, but in summary, they found out that ArcFace yields the best results, compared to SphereFace, CosFace and even to the combination of them.

They also confirmed experimentally that ArcFace perform better than adding terms such as Intra-Loss, Inter-Loss and Triplet-Loss, being the best at minimizing intra-class distance while at the same time maximizing inter-class distances. The other terms may be a little bit better on one of the two tasks, but impacting negatively at the other task, making it worse overall.

Finally, on top of that, they improved the results of the state-of-the-art in several benchmarks, such as LFW, YTF, CALFW, CPLFW, MegaFace, and others

3.1.3. Spoofing

A complementary problem to the recognition problem is spoofing. Spoofing is the attempt to deceive the system into falsely recognizing or verifying an identity. Given that these systems are sometimes used as biometric safety measures (unlocking a laptop or computer), it is imperative to study which is the extent of the security provided, and in which cases the system can be misled into false positives. At the same time, research in this area has shed light on the recognition problem as a whole.

There are different techniques to spoof a system, these range from showing the camera capturing the image a photograph of the person, to 3D printed masks, with varying results. To defend a system from these attacks, there are several techniques that can be employed, for example: liveness detection, challenge-response, 3D cameras, etc. In the following Chapter, we will propose a new approach that by exploiting 3D information present in a 2D-captured image, virtually any 2D attack can be prevented [17]. At the same time, the need for a 3D camera and performing real 3D reconstruction is avoided. This is desirable, since 3D cameras are usually expensive and hard to use.

3.2. Active Stereo Geometry

When a structured pattern of light is projected over a surface, it is perceived with a certain deformation depending on the shape of the surface. For example, the top left image in Figure 4.5 is acquired while horizontal stripes are projected over the subject face. As we can see, fringes are no longer perceived as horizontal and parallel to each other, this deformation codes rich information about the geometry of the scene, which will be exploited to enhance 2D face recognition methods.

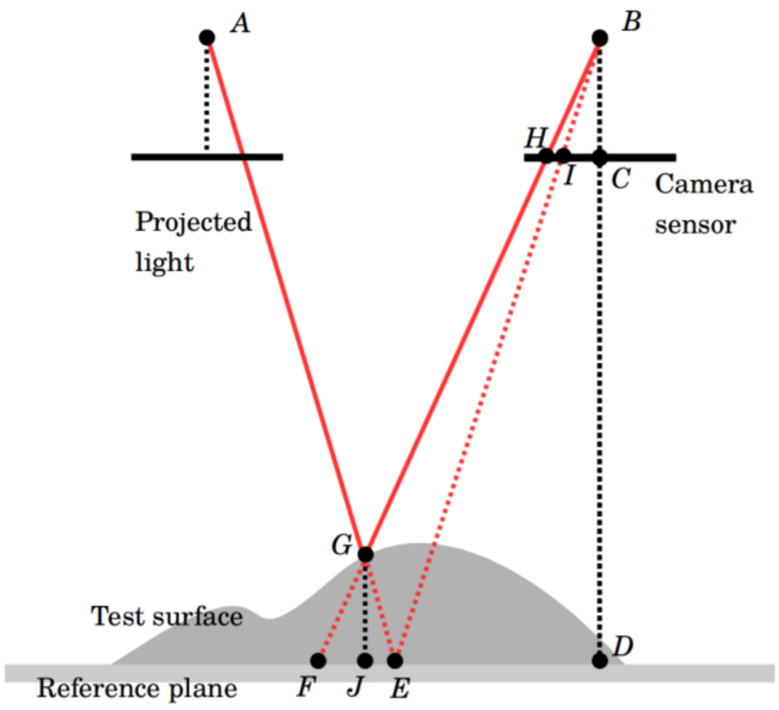


Figure 3.8: Active stereo geometry. Left sensor illustrates the optical image plane of the device projecting light (with optical center at point A), and the right sensor illustrates the camera sensor (with optical center at B). A ray of light projected over a reference plane (at point E) is perceived by the camera at the pixel position I. When the same ray of light is projected over an arbitrary surface (at point G), it is perceived by the camera at a shifted location H. The disparity $\overline{IH}$ is proportional to $\overline{JG}$ (height of the surface).

Figure 3.8 sketches the geometry of this situation as if we were looking at the face from the side (vertical section). On the left, a light source project a ray of light through the points AG (red line). When this ray is reflected by a reference plane at the point E, it is *viewed* by the camera at the pixel location represented by I. If instead, light is projected over an arbitrary (non-planar) surface, the reflection is produced at point G and viewed by the camera at the shifted position H. From now on, we denote the length of the shift $\overline{IH}$ as the disparity ϕ.

Similarity between triangles $\widehat{FBE}$-$\widehat{HBI}$ and $\widehat{FGE}$-$\widehat{AGB}$ leads to $\overline{EF}/\overline{DB} = \overline{IH}/\overline{CB}$ and $\overline{FE}/\overline{JG} = \overline{AB}/(\overline{DB} - \overline{JG})$. Defining $\overline{AB} = b$ (baseline between the lighting source and the camera sensor), $\overline{BC} = f$ (camera focal length), $\overline{JG} = z$ (surface local height), $\overline{BD} = q$ (distance of the subject to the camera), and assuming that $q \gg f, z$ we obtain,

$$\phi = z\frac{bf}{q^2} \Rightarrow \nabla\phi \propto \nabla z. \tag{3.5}$$

Equation (3.5) quantitatively relates the perceived shift (disparity) d of the projected pattern and the surface 3D shape z. As the goal of the present work is to provide local 3D features instead of performing an actual 3D reconstruction, the particular value of b, f and h are irrelevant as we shall see. Moreover, we exploit the fact that the gradient of the local disparity codes information of the depth gradient. This allows us to extract local geometrical features bypassing the more challenging steps of 3D reconstruction: global matching and gradient field integration [18, 30, 72, 108].

Esta página ha sido intencionalmente dejada en blanco.

Chapter 4

3D Face Recognition without depth reconstruction nor generic 3D models

In this Chapter, we will focus on a particular research topic: **3D face recognition without depth reconstruction nor generic 3D models**. We look into the work done on this subject that was published in IEEE TPAMI [17] and presented as a full paper in the International Conference in Computational Photography 2020 [68].

4.1. Abstract

Active illumination is a prominent complement to enhance 2D face recognition and make it more robust, e.g., to spoofing attacks and low-light conditions. In the published work, we show that it is possible to adopt active illumination to enhance state-of-the-art 2D face recognition approaches with 3D features, while bypassing the complicated task of 3D reconstruction. The key idea is to project over the test face a high spatial frequency pattern, which allows us to simultaneously recover real 3D information plus a standard 2D facial image. Therefore, state-of-the-art 2D face recognition solutions can be transparently applied, while from the high frequency component of the input image, complementary 3D facial features are extracted. Experimental results on ND-2006 dataset show that the proposed ideas can significantly boost face recognition performance and dramatically improve the robustness to spoofing attacks.

Two-dimensional face recognition has become extremely popular as it can be ubiquitously deployed and large datasets are available. In the past several years, tremendous progress has been achieved in making 2D approaches more robust and useful in real-world applications. Though 2D face recognition has surpassed human performance in certain conditions, challenges remain to make it robust to facial poses, uncontrolled ambient illumination, aging, low-light conditions, and spoofing attacks [19, 42, 66, 90]. We address some of these issues by enhancing the captured RGB facial image with 3D information, as illustrated in Figure 4.1.

4.2. Introduction

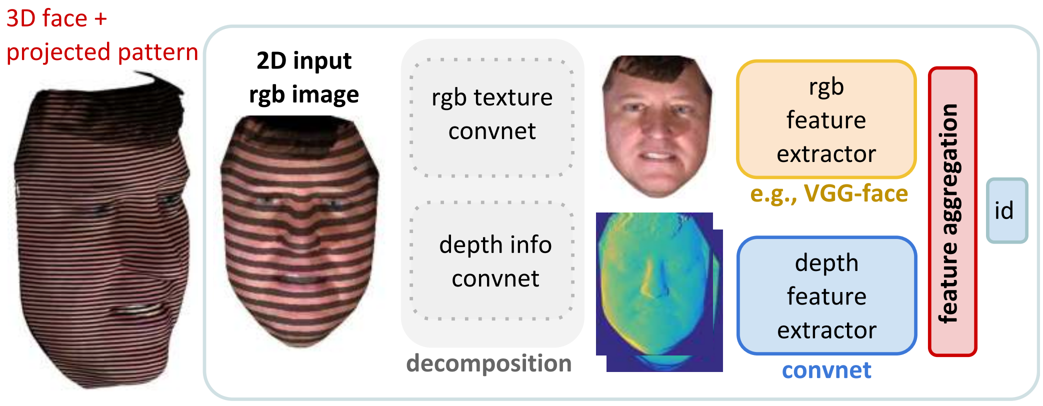


Figure 4.1: Real 3D face recognition is possible by capturing one single RGB image if a high frequency pattern is projected. The low frequency components of the captured image can be fed into a state-of-the-art 2D face recognition method, while the high frequency components encode local depth information that can be used to extract 3D facial features. It is important to highlight that, in contrast with most existing 3D alternatives, the proposed approach provides real 3D information, not 3D hallucination from the RGB input. As a result, state-of-the-art 2D face recognition methods can be enhanced with real 3D information.

High resolution cameras became ubiquitous, however, for 2D face recognition, we only need a facial image of moderate or low resolution. For example latest phones frontal camera have a very high resolution (e.g., 3088×2320 pixels) while the resolution of the input to most face recognition systems is limited to 224×224 pixels [14,69,77,90,117]. This means that, in the context of face recognition, we are drastically underutilizing most of the resolution of captured images. We propose an alternative to use the discarded portion of the spectra and extract real 3D information by projecting a high frequency light pattern. Hence, a low resolution version of the RGB image remains approximately invariant allowing the use of standard 2D approaches, while 3D information is extracted efficiently from the local deformation of the projected patterns.

The proposed solution to extract 3D facial features has some key differences with the two common approaches presented in existing literature: 3D hallucination [24, 39, 56, 70] and 3D reconstruction [104, 116]. We will discuss these differences in detail in the following Section. We illustrate the main limitation of 3D hallucination in the context of face recognition in Figure 4.2, which emphasizes the lack of real 3D information on a standard RGB input image. We demonstrate that it is possible to extract actual 3D facial features bypassing the ill-posed problem of explicit depth estimation.

The contributions made by the paper are summarized as follows:

- Analyzing the spectral content of thousands of facial images, we design a high frequency light pattern that simultaneously allow us to retrieve a standard 2D low resolution facial image plus a 3D gradient facial representation.

- We propose an effective and modular solution that achieves 2D and 3D information decomposition and facial feature extraction in a data-driven fashion (bypassing a 3D facial reconstruction).
- We show that by defining an adequate distance function in the space of the feature embedding, we can leverage the advantages of both 2D and 3D features. We can transparently exploit existing state-of-the-art 2D methods and improve their robustness, e.g., to spoofing attacks.

4.3. Related Work

To recognize or validate the identity of a subject from a 2D color photograph is a longstanding problem of computer vision and has been largely studied for over forty years [41, 113]. Recent advances in machine learning, and in particular, the success of deep neural networks, reshaped the field and yielded more efficient, accurate, and reliable 2D methods such as: ArcFace [14], VGG-Face [69], DeepFace [90], and FaceNet [77].

In spite of this, spoofing attacks and variations in pose, expression and illumination are still active challenges and significant efforts are being made to address them [10, 32, 35, 46, 51, 59, 93, 103, 112, 116]. For example, Deng et al. [13] attempt to handle large pose discrepancy between samples. To that end, they propose an adversarial facial UV map completion GAN. Complementing previous approaches that seek for robust feature representations, several works propose more robust loss and metric functions [57, 97].

4.3.1. 3D hallucination from single RGB.

To enhance 2D approaches, a common trend is to hallucinate a 3D representation from an input RGB image which is used to extract 3D features [8, 20, 24, 39, 56, 70].

For example, Cui et al. [12] introduce a cascade of networks that simultaneously recover depth from an RGB input while seeking for separability of individual subjects. The estimated depth information is then used as a complementary modality to RGB.

Limitations of 3D hallucination in the context of face recognition. As discussed in Section 4.3 3D hallucination methods have intrinsic limitations in the context of face recognition. To complement the example illustrated in Figure 4.2, here we show 5 3D facial models obtained by hallucinating 3D from a single RGB input image. To that end, we apply the 3D morphable model (extremely popular in facial applications). As we can see, even in the case of a planar spoofing attack, a face-like 3D shape is retrieved despite that this is far form the actual 3D shape of the actual scene. This is an expected result, as we discuss in Section 4.3 the problem of 3D hallucination is ill-posed, and therefore priors need to be enforced in order to obtain feasible implementations.

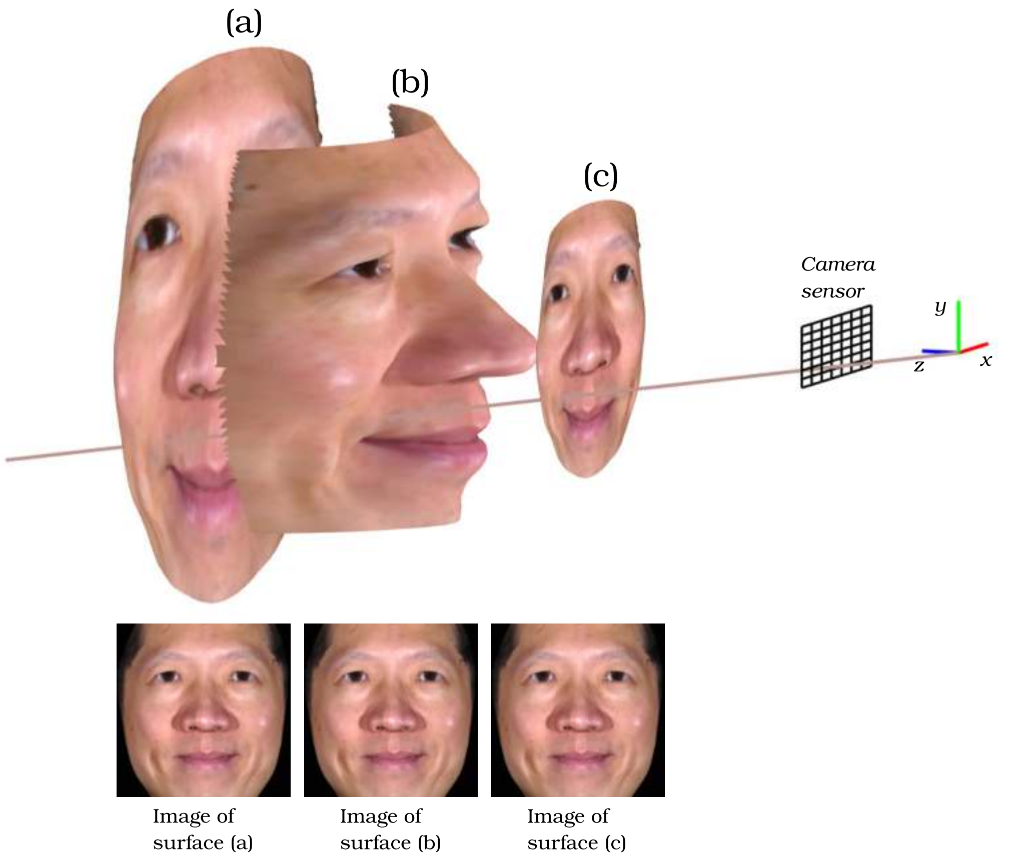


Figure 4.2: Illustration of three different 3D surfaces that look equivalent from a monocular view (single RGB image). On top, three surfaces (a), (b) and (c) are simulated, being (a) and (c) flat and (b) the 3D shape of a test subject. We use classic projective geometry [30] and simulate the image we obtain when photographing (a), (b) and (c) respectively. The resulting images are shown at the bottom. As we illustrate with this simple example, the relation between images and 3D scenes is not bijective and the problem of 3D hallucination is ill-posed. To overcome this, 3D hallucination solutions enforce important priors about the geometry of the scene. This is why we argue that these methods do not really add to the face recognition task, actual 3D information. (A complementary example is presented in Figure 4.3.)

4.3.2. 3D face reconstruction.

The approaches described previously share an important practical advantage that at the same time is their weakness, they extract all the information from a standard (RGB) 2D photograph of the face. As depicted in Figure 4.2 a single image does not contain actual 3D information.

To overcome this intrinsic limitation different ideas have been proposed and datasets with 3D facial information are becoming more popular [117]. For example, Zafeiriou et al. [104] propose a four-light source photometric stereo (PS). A similar idea is elaborated by Zou et al. [116] who propose to use active near-infrared illumination and combine a pair of input images to extract an illumination invariant

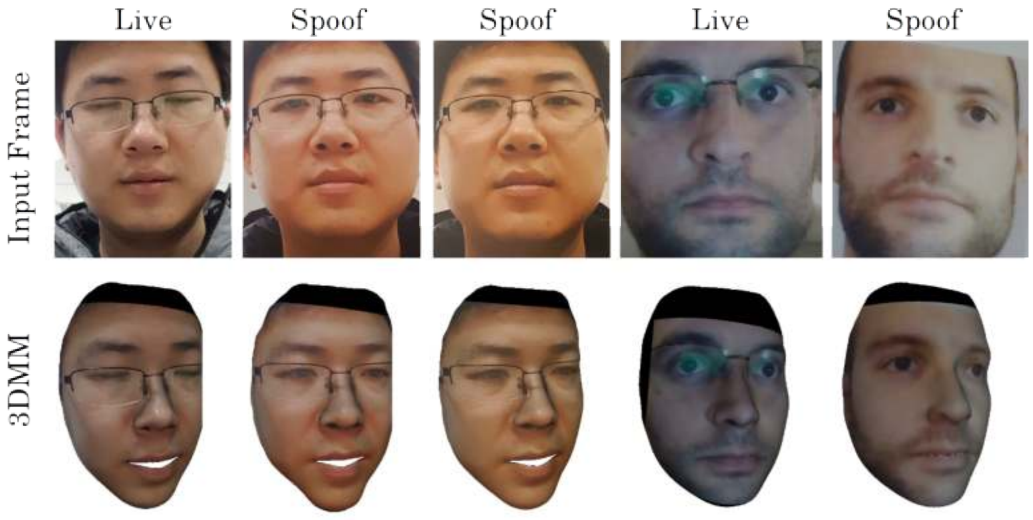


Figure 4.3: Example of 3D hallucination [8] from photographs of live faces versus portraits. Second row illustrates the 3D morphable model (second row) of images of live subjects (first and fourth columns) and for photographs of portraits of the same subjects (second, third and fifth columns). Results obtained using the code from [39]

face representation.

Despite the previous mentioned techniques, performing a 3D facial reconstruction is still a challenging and complicated task. Many strategies have been proposed to tackle this problem, including time delay based [61], image cue based [23, 24, 48, 71, 76], and triangulation based methods [4, 16, 52, 75, 109].

Although there has been great recent development, available technology for 3D scanning is still too complicated to be ubiquitously deployed [18, 30, 107, 108].

The proposed solution has two key features that make it, to the best of our knowledge, different from existing alternatives. (a) Because the projected pattern is of a high spatial frequency, we can recover a standard (low resolution) RGB facial image that can be fed into state-of-the-art 2D face recognition methods. (b) We avoid the complicated task of 3D facial reconstruction and instead, extract local 3D features from the local deformation of the projected pattern. In that sense our ideas can be implemented exploiting existing and future 2D solutions. In addition, our approach is different from those that hallucinate 3D information. As discussed before and illustrated in Figure 4.2 this task requires a strong prior of the scene which is ineffective, for example, if a spoofing attack is presented (Figure 4.3).

4.4. Gradient Information is Easy to Compute, Absolute Depth is Hard.

One of the key ideas of the presented approach is to estimate local depth descriptors instead of absolute depth information. While the former is an easier task, absolute information is irrelevant for the sake of feature extraction. After all, a robust feature representation should be scale and translation invariant.

More precisely, we can define the problem of integrating the absolute depth z from an empirical estimation of its gradient map $(\tilde{z_x}, \tilde{z_y})$ in a 2D domain $\Omega \in \mathbb{R}^2$ as the optimization problem:

$$z^* = \text{argmin}_{z \in \mathcal{S}} \iint\limits_{\Omega} \|\nabla z - (\tilde{z_x}, \tilde{z_y})^T\| \, \partial\Omega. \tag{4.1}$$

The solution of Equation (4.1) is non trivial. Noise, shadows, and facial discontinuities produce empirical gradient estimations $(\tilde{z_x}, \tilde{z_y})$ with irotational components, i.e., for some pixels (x_i, y_i) the rotor of the field is different from zero $\nabla \times (\tilde{z_x}(x_i, y_i), \tilde{z_y}(x_i, y_i))^T \neq 0$. Therefore, the space of target functions $\mathcal{S}$ and the minimization norm $\| \cdot \|$ must be carefully set in order to achieve a meaningful solution to Equation (4.1). This is a complex mathematical problem and has been extensively studied in the literature [1, 2, 21, 73, 94].

4.5. Proposed Approach

Notation. Let $\mathcal{I} \subset \mathbb{R}^{H \times W \times C}$ denote the space of images with $H \times W$ pixels and C color channels, and $\mathcal{X}_n \subset \mathbb{R}^n$ a space of n-dimensional column vectors (in the context of this work associated to a facial feature embedding).

$\mathcal{I}_{rgb}$ denotes the set of RGB images ($C = 3$), while $\mathcal{I}_{\nabla z}$ is used to denote the space of two channel images ($C = 2$) associated to the gradient of a single-channel image $z \in \mathbb{R}^{H \times W \times 1}$. The first/second channel represents the partial derivative with respect to the first/second coordinate.

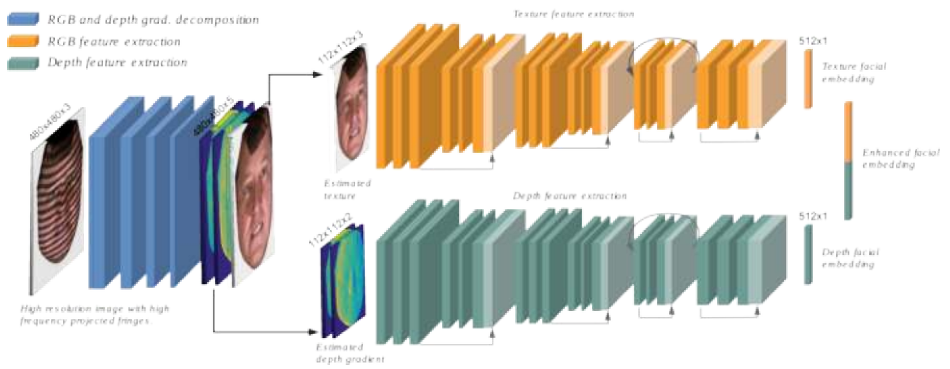


Figure 4.4: Architecture overview. First a network (illustrated in blue) is used to decompose the input image that contains overlapped high frequency fringes into a lower resolution (standard) texture facial image and depth gradient information. The former is used as the input of a state-of-the-art 2D face recognition DNN (yellow blocks). Depth information is fed to another network (green blocks) trained to extract discriminative (depth-based) facial features. Different network architectures were tested, we provide implementation details in Section 4.7.

Combining depth and RGB information. The proposed approach consists of three main modules as illustrated in Figure 4.4:

- $g : \mathcal{I}_{rgb} \rightarrow \mathcal{I}_{rgb} \times \mathcal{I}_{\nabla z}$ performs a decomposition of the input image into texture and depth information.
- $f_{rgb} : \mathcal{I}_{rgb} \rightarrow \mathcal{X}_{n/2}$ extract facial features associated to the facial texture.
- $f_{\nabla z} : \mathcal{I}_{\nabla z} \rightarrow \mathcal{X}_{n/2}$ extract facial features associated to the facial depth.

These three components are illustrated in Figure 4.4 in blue, yellow, and green, respectively. We decided to have three modules instead of a single *end-to-end* design for several reasons that will be discussed below.

We denote the facial feature extraction from the input image as $f_\theta : \mathcal{I}_{rgb} \rightarrow \mathcal{X}_n$, where $f_\theta(I) = (f_{rgb}(I_{rgb}), f_{\nabla z}(I_{\nabla z}))^T$ with $\{I_{rgb}, I_{\nabla z}\} = g(I)$. The subscript θ represent the parameters of the mapping f, which can be decomposed in three groups $\theta = (\theta_g, \theta_{rgb}, \theta_{\nabla z})$, associated to the image decomposition, RGB feature extraction, and depth feature extraction respectively. In the following we discuss how these parameters are optimized for each specific task, which is one of the advantages of formulating the problem in a modular fashion.

Once texture and depth facial information is extracted into a suitable vector representation $x = f_\theta(I)$ (as illustrated in Algorithm 1), we can select a distance measure $d : \mathcal{X}_n \times \mathcal{X}_n \rightarrow \mathbb{R}^+$ to compare facial samples and estimate whether they have a high likelihood of belonging to the same subject or not. This process is illustrated in algorithm 2.

It is worth noticing that faces are embedded into a space in which the first half of the dimensions are associated to information extracted from the RGB representation while the other half codes depth information. These two sources of information may have associated different confidence levels (depending on the conditions at deployment). We address this in detail in Section 4.5.3 and propose an anisotropic distance adapted to our solution, and capable of leveraging the good performance of 2D solutions in certain conditions, while improving robustness and handling spoofing attacks in a continuous and unified fashion.

Algorithm 1 Compute 2D facial features enhanced with 3D information.

```
1: procedure FacialEmbedding($I$)
   Decompose the input image into texture and depth gradient information.
2:     $\{I_{rgb}, I_{\nabla z}\} = g(I)$
   Extract facial information from each component.
3:     $x_{rgb} = f_{rgb}(I_{rgb})$
4:     $x_{\nabla z} = f_{\nabla z}(I_{\nabla z})$
   Combine texture and depth information.
5:     $x = \text{Concatenate}(x_{rgb}, x_{\nabla z})$
6:     return $x$                                  ▷ Facial embedding
7: end procedure
```

Algorithm 2 The two most common cases of use: "Subject identification" is to find the id candidates of a sample x by finding its k nearest neighbors on a predefined gallery X. "Subject verification" verifies if two representations x and $x_{template}$ belong to the same subject by comparing their distance with respect to a predefined threshold ϵ.

```
1: procedure SubjectIdentification($x$,$X$)
   Compare subject embedding $x$ against the dataset $X$
2:     ids = argsort(distance($x, X[i]$))
3:     return $ids[: k]$                                  ▷ $k$ candidate ids.
4: end procedure
5: procedure SubjectVerification($x$,$x_{template}$)
   Compare the sample $x$ against the template $x_{template}$
6:     idVerified = distance($x, x_{template}$) $< \epsilon$
7:     return idVerified
8: end procedure
```

4.5.1. Pattern design.

When a pattern of light $p(x, y)$ is projected over a surface with a height map $z(x, y)$, it is perceived by a camera located along the x-axis with a deformation given by $p(x+\phi(x, y), y)$ ($\phi(x, y) \propto z(x, y)$) (see Section 3.2). Let us denote $I_0(x, y)$ the image we would acquire under homogeneous illumination, and $p(x, y)$ the intensity profile of the projected light. Without loss of generality we assume the system baseline is parallel to the x axis. The image acquired by the camera when the projected light is modulated with a profile $p(x, y)$ is

$$I(x, y) = I_0(x, y)p(x + \phi(x, y), y). \tag{4.2}$$

We will restrict to periodic modulation patterns and let T denote the pattern spatial period, we also define $f_0 \overset{def}{=} \frac{1}{T}$. To simplify the system design and analysis, lets also restrict to periodic patterns that are invariant to the y coordinate. In these conditions we can express $p(x, y) = \sum_{n=-\infty}^{+\infty} a_n \, e^{i2\pi n f_0 x}$ where a_n represent the coefficients of the Fourier series of p. Note that because of the invariance with respect to the y coordinate, the coefficients a_n are constant instead of a function of y.

Equation (4.2) can be expressed as

$$I(x, y) = \sum_{n=-\infty}^{+\infty} I_0(x, y) \, a_n \, e^{i2\pi n f_0 (x+\phi(x,y))}. \tag{4.3}$$

Defining $q_n(x, y) \overset{def}{=} I_0(x, y) \, a_n \, e^{i2\pi n f_0 \phi(x,y)}$, Equation (4.3) can be expressed as [91]

$$I(x, y) = \sum_{n=-\infty}^{+\infty} q_n(x, y) e^{i2\pi n f_0 x}. \tag{4.4}$$

Applying the 2D Fourier Transform (FT) in both sides of Equation (4.4) and using standard properties of the FT [78] we obtain

$$\tilde{I}(f_x, f_y) = \sum_{n=-\infty}^{+\infty} \tilde{q_n}(f_x - nf_0, f_y). \tag{4.5}$$

We denote as $\tilde{I}$ the FT of I and use (f_x, f_y) to represent the 2D frequency domain associated to x and y axis respectively.

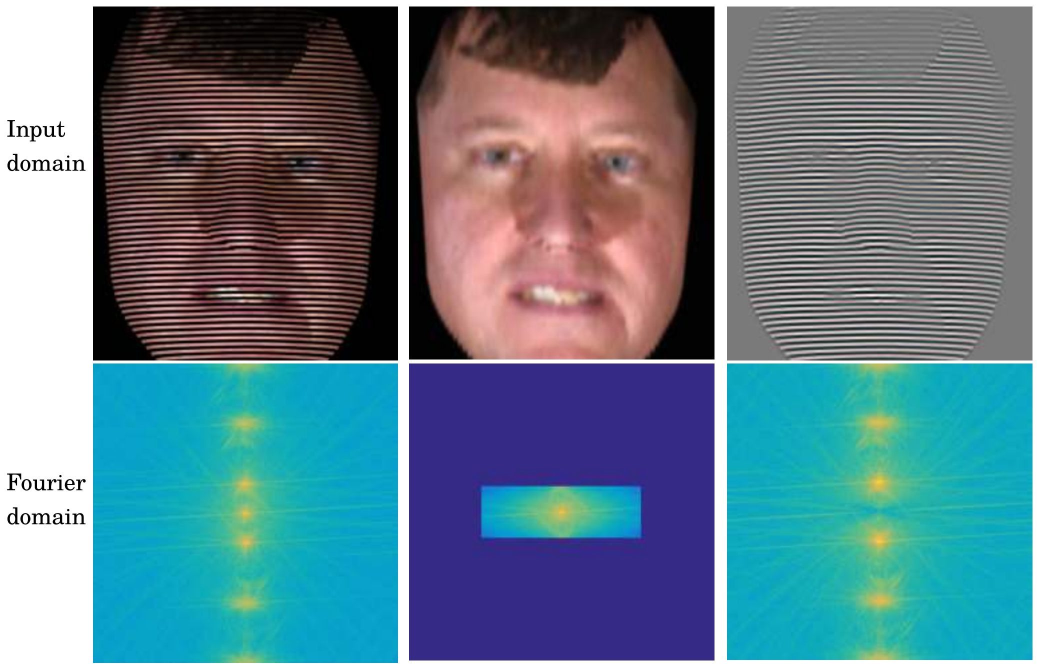


Figure 4.5: 2D plus real 3D in a single rgb image. The first column illustrates the RGB image acquired by a (standard) camera when horizontal stripes are projected over the face. The second column isolates the low frequency components of the input image, and the third column corresponds to the residual high frequency components (In all the cases the absolute value of the Fourier Transform is represented in logarithmic scale). As can be seen, high frequency patterns can be used to extract 3D information of the face (third column) while preserving a lower resolution version of the facial texture (middle column).

Equation (4.5) shows that the FT of the acquired image can be decomposed into the components $\tilde{q_n}$ centered at $(nf_0, 0)$. In the context of this section, we refer to a function $h(x, y)$ being smooth if

$$\frac{\|\tilde{h}(f_x, f_y)\|}{\|\tilde{h}(0,0)\|} < 10^{-3} \quad \forall \quad |f_x| > \frac{f_0}{2}. \tag{4.6}$$

Assuming $I_0(x, y)$ and $\phi(x, y)$ are smooth (we empirically validate this hypothesis below), the components $\tilde{q_n}$ can be isolated as illustrated in Figure 4.5.

The central component is of particular interest, $q_0(x, y) = a_0 \, I_0(x, y)$ as it captures the facial texture information and can be recovered from $I(x, y)$ if f_0 is large enough (we provide a more precise quantitative analysis in what follows). On the other hand, relative (gradient) 3D information can be retrieved from the components $\{q_0, q_1\}$ as we show in Proposition 1.

Proposition 1. Gradient depth information is encoded in the components $\{q_0(x,y), q_1(x,y)\}$.

Proof. We define the wrapping function $\mathcal{W}(u) = \text{atan}(\tan(u))$. This function wraps the real set into the interval $(-\pi/2, \pi/2]$ [72]. This definition can be extended to vector inputs wrapping the modulus of the vector field while keeping its direction unchanged, i.e., $\mathcal{W}(\vec{u}) = \frac{\mathcal{W}(\|\vec{u}\|)}{\|\vec{u}\|}\vec{u}$ if $\|\vec{u}\| \neq 0$ and $\mathcal{W}(\vec{u}) = \vec{0}$ if $\|\vec{u}\| = 0$.

From $q_1(x,y)$ and $q_0(x,y)$ we can compute[1]

$$\phi_{\mathcal{W}}(x,y) = \frac{1}{2\pi f_0}\,\text{atan}\left(\frac{\text{Im}\{\frac{q_1(x,y)}{q_0(x,y)}\}}{\text{Re}\{\frac{q_1(x,y)}{q_0(x,y)}\}}\right) \tag{4.7}$$

where $\phi_{\mathcal{W}}$ denotes the wrapped version of ϕ. Moreover, $\phi_{\mathcal{W}}(x,y) = \phi(x,y) + \pi k(x,y)$ with $k(x,y) \in \mathbb{N}$ (wrapping introduces shifts of magnitude multiple of π). Computing the gradient both sides leads to $\nabla\phi_{\mathcal{W}}(x,y) = \nabla\phi(x,y) + \pi\nabla k(x,y)$ where $\|\nabla k(x,y)\| \in \mathbb{N}$. Assuming the magnitude of the gradient of $\phi(x,y)$ is bounded by $\pi/2$ and considering that $\|\nabla k(x,y)\| \in \mathbb{N}$, we can apply the wrapping function both sides of the previous equality to obtain $\mathcal{W}(\nabla\phi_{\mathcal{W}})(x,y) = \nabla\phi(x,y)$ which proves (recall Equation (4.7)) that the gradient of ϕ can be extracted from the components q_0 and q_1. To conclude the proof, we use the property of linearity of the gradient operation and the fact that $\phi(x,y)$ is proportional to the depth map of the scene (see Equation (3.5) and Section 3.2).

Analytic versus data-driven texture and gradient depth extraction The previous analysis shows that closed forms can be obtained to extract texture and depth gradient information. However, to compute these expressions is necessary to isolate different spectral components $\tilde{q_n}$. To that end, filters need to be carefully designed. The design of these filters is challenging, e.g., one need to control over-smoothing versus introducing ringing artifact which are drastically amplified by a posterior gradient computation [16,109]. To overcome these challenges, we chose to perform a depth (gradient) and texture decomposition in a data-driven fashion, which as we show in Section 4.6, provides an efficient and effective solution.

Bounds on f_0 and optimal spectral orientation. As discussed above, the projected pattern $p(x,y)$ should have a large fundamental frequency f_0. In addition, the orientation of the fringes and the system baseline can be optimized if faces present a narrower spectral content in a particular direction. We study the texture and depth spectrum of the facial images of ND-2006 dataset (this dataset provides ground truth facial texture and depth information).

[1]We assume images are extended in an even fashion outside the image domain, to guaranteed that $a_1 \in \mathbb{R}$ and avoid an additional offset term.

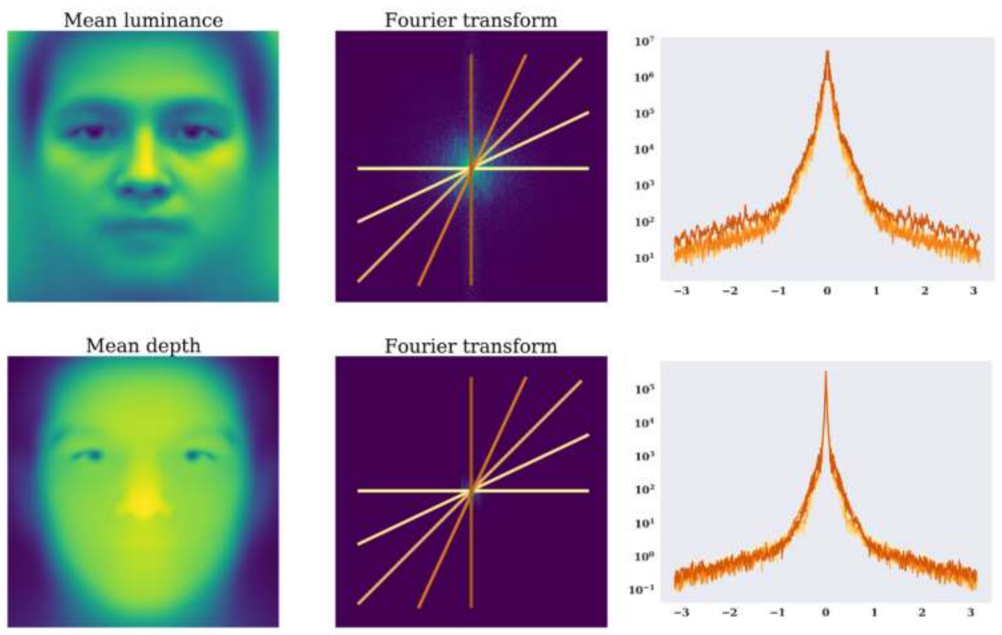


Figure 4.6: Faces average spectral content. The first column illustrates the mean luminance and depth map for the faces in the dataset ND-2006. The second column shows the mean Fourier Transform of the faces luminance and depth respectively. The third column shows the profile across different sections of the 2D Fourier domain. Columns two and three represent the absolute value of the Fourier transform in logarithmic scale. Faces are registered using the eyes landmarks and the size normalized to 480×480 pixels.

We observed (see Figure 4.6) that for facial images sampled at a 480×480 spatial resolution, most of the energy is concentrated in a third of the discrete spectral domain (observe the extracted one dimensional profiles of the spectrum shown at the left side of Figure 4.6). In addition, we observe that the spectral content of facial images is approximately isotropic. See, for example, Figure 4.6 and observe how for 1-dimensional sections across different orientations the 2D spectra envelope is almost constant. We conclude that the orientation of the fringes does not play a significant role in the context of facial analysis. In addition, we conclude that the fringes width should be smaller than 7mm (distance measured over the face).[2]

4.5.2. Network training and the advantages of modularity.

As described previously, the parameters of the proposed solution can be split in three groups $\theta = (\theta_g, \theta_{rgb}, \theta_{\nabla z})$. This is an important practical property and we designed the proposed solution to meet this condition (in contrast to an end-to-end

[2]This numerical result is obtained by approximating the bounding box of the face as a $20cm \times 20cm$ region, sampled with 480×480 pixels which corresponds to a pixel length of $2{,}4mm$, a third of the spectral band correspond to signal of a period of 6 pixels which leads to a binary fringe of at least $7{,}2mm$ wide.

approach).

Let us define $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ three datasets containing ground truth depth information, ground truth identity for rgb facial images, and ground truth identity for depth facial images, respectively. More precisely, $\mathcal{B}_1 = \{(I_i(x,y), I_{0i}(x,y), z_i(x,y)),\ i = 1, ..., n_1\}$, $\mathcal{B}_2 = \{(I_{0i}(x,y), y_i),\ i = 1, ..., n_2\}$, and $\mathcal{B}_3 = \{(z_i(x,y), y_i),\ i = 1, ..., n_3\}$, where $I_i(x,y)$ denotes a (facial or generic) RGB image acquired under the projection of the designed pattern, $I_{0i}(x,y)$ represents (facial or generic) standard RGB images, $z_i(x,y)$ denotes a gray image representing the depth of the scene, and y_i a scalar integer representing the subject id.

We denote as $\{g_1(I), g_2(I)\} = g(I)$ the RGB and gradient depth components estimated by the decomposition operation g. We partitioned the parameters of g into two sets of dedicated kernels $\theta_g = \{\theta_{g1}, \theta_{g2}\}$, the first group focuses on retrieving the texture component while the second group retrieves the depth gradient. These parameters can be optimized as

$$\theta_{g1} = \operatorname{argmin} \sum_{(I_{0i}, I_i) \in \mathcal{B}_1} \|g_1(I_i) - I_{0i}\|_2^2 \tag{4.8}$$

$$\theta_{g2} = \operatorname{argmin} \sum_{(z_i, I_i) \in \mathcal{B}_1} \|g_2(I_i) - \nabla z_i\|_2^2. \tag{4.9}$$

We also evaluated training a shared set of kernels trained with an unified loss. This alternative is harder to train in practice, due to the natural difference between the dynamic range and sparsity of gradient images compared with texture images.

For texture and depth facial feature extraction, we tested models inspired in the Xception architecture [11]. Additional details are provided in Section 4.7. To train these models we add an auxiliary fully connected layer on top of the facial embedding (with as many neurons as identities in the train set) and minimize the cross-entropy between the ground truth and the predicted labels. More precisely, let us denote $\hat{f}_{rgb}(I_{rgb}) = [p_1, ..., p_c]$ the output of the fully connected layer associated to the embedding $f_{rgb}(I_{rgb})$ where p_i denotes the probability associated to the id i,

$$\theta_{rgb} = \operatorname{argmin} \sum_{(I_{0i}, y_i) \in \mathcal{B}_2} \sum_{c} -\mathbf{1}_{y_i = c} \log(\hat{f}_{rgb}(I_{0i})[c]) \tag{4.10}$$

$$\theta_{\nabla z} = \operatorname{argmin} \sum_{(z_i, y_i) \in \mathcal{B}_3} \sum_{c} -\mathbf{1}_{y_i = c} \log(\hat{f}_{\nabla z}(\nabla z_i)[c]) \tag{4.11}$$

where $\mathbf{1}_{y_i = c}$ denotes the indicator function. Of course, one can choose other alternative losses to train these modules, see e.g., [14, 57, 97, 114].

As described above, the proposed design allows to leverage information from three types of datasets ($\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$). This has an important practical advantage as 2D facial and 3D generic datasets are more abundant, and the pattern dependant set $\mathcal{B}_1$ can be of modest size as $\#(\theta_g) \ll \#(\theta_{rgb})$.

4.5.3. Distance design.

Once different modules are set we can compute the facial embedding of test images following the procedure described in Algorithm 1. Let us define $x^a \in \mathcal{X}_n$ and $x^b \in \mathcal{X}_n$ the feature embedding of two facial images I_a and I_b respectively. Recall that the first $n/2$ elements of x are associated to features extracted from (a recovered) RGB facial image while the remaining elements are associated to depth information, i.e., $x = (x_{rgb}[1], ..., x_{rgb}[n/2], x_{\nabla z}[1], ..., x_{\nabla z}[n/2])^T$.

We define the distance between two feature representations $x^a = (x^a_{rgb}, x^a_{\nabla z})$, and $x^b = (x^b_{rgb}, x^b_{\nabla z})$ as

$$\begin{aligned} d_{\alpha,\beta,\gamma}(x^a, x^b) &\overset{def}{=} (1-\gamma)\, d_c(x^a_{rgb},\, x^b_{rgb}) \\ &+ \gamma\, d_c(x^a_{\nabla z},\, x^b_{\nabla z}) \left(1 + \left(\frac{d_c(x^a_{\nabla z}, x^b_{\nabla z})}{\beta}\right)^{\alpha}\right). \end{aligned} \tag{4.12}$$

$d_c : \mathcal{X}_{n/2} \times \mathcal{X}_{n/2} \to [0, 1]$ denotes the cosine distance, $\gamma \in [0, 1]$ sets the relative weight of RGB and depth features, and $\alpha, \beta \in \mathbb{R}$ define a non-linear response for the distance between depth features. As we will describe in the following, this provides robustness against common cases of spoofing attacks.

Intuitively, γ allows us to set the relative confidence associated to RGB and depth features, for example, $\gamma = 1/2$ gives the same weight to RGB and depth features, while $\gamma = 0$ ($\gamma = 1$) ignores the distance between samples in the depth (RGB) embedding space. This is important in practice, as is common to obtain substantially more data to train RGB models than depth ones ($|\mathcal{B}_2| \gg |\mathcal{B}_3|$). This suggests that in good test conditions (e.g., good lighting) one may trust more RGB features over depth features ($\gamma < 1/2$). As we will empirically show in the following Section, when two facial candidates are compared, $d_{\alpha,\infty,\gamma}(x^a, x^b) = (1-\gamma)\, \delta(x^a_{rgb},\, x^b_{rgb}) + \gamma\, \delta(x^a_{\nabla z},\, x^b_{\nabla z})$ is an effective distance choice. However, it does not handle robustly common cases of spoofing attacks. The most common deployments of spoofing attacks imitate the facial texture more accurately than the facial depth [9, 58, 110], therefore, the global distance between two samples should be large when the distance of the depth features is large (i.e. above a certain threshold). To that end, we introduce an additional non-linear term controlled by parameters β and α, for $\delta(x^a_{\nabla z}, x^b_{\nabla z}) < \beta$ the standard cosine distance dominates while for large values the distance will be amplified in a non-linear fashion.

4.6. Experiments and Discussion

Data. Three public dataset are used for experimental validation: FaceScrub [67], CASIA Anti-spoofing [111], and ND-2006 [25]. FaceScrub contains $100k$ RGB (2D) facial images of 530 different subjects, and is used to train the texture-based facial embedding. CASIA dataset contains 150 genuine videos (recording a person) and 450 videos of different types of spoofing attacks, the data was collected for 50 subjects. We use this dataset to simulate and imitate the texture properties of images of spoofing attacks. ND-2006 is one of the larges publicly available datasets

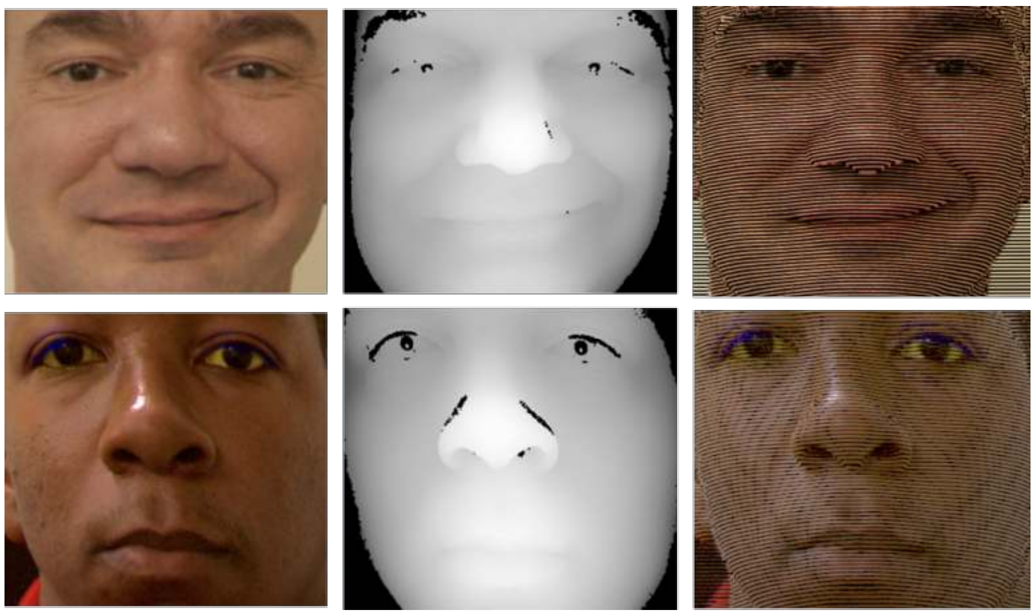


Figure 4.7: Active light projection. From left to right: ground truth RGB facial image, 3D facial scanner, and finally the image we would acquire if the designed high frequency pattern is projected over the face. Two random samples from ND-2006 are illustrated.

with 2D and 3D facial information, it contains $13k$ images of 888 subjects. We used this set to demonstrate that differential 3D features can be extracted from a single RGB input, to compare RGB features with 3D features extracted from the differential 3D input, and to show that when 2D and 3D information is properly combined, the best properties of each can be obtained.

Texture and differential 3D decomposition. In Section 4.5.1 we discussed how real 3D information and texture information can be coded and later extracted using a single RGB image. In addition, we argue that this decomposition can be learned efficiently and effectively in a data-driven fashion.

To that end, we tested simple network architectures composed of standard convolutional layers (a full description of these architectures and the training protocols are provided in Section 4.7). Using ground truth texture and depth facial information, we simulated the projection of the designed pattern over the 888 subjects provided in ND-2006 dataset. Illustrative results are presented in Figure 4.7. The 3D geometrical model and a detailed description of the simulation process is provided in Section 4.7.1. Though the simulation of the deformation of a projected pattern can be computed in a relatively simple manner (if the depth information is known), the inverse problem is analytically hard [16, 75, 108].

Despite the previous, we observed that a stack of convolutional layers can efficiently learn how to infer from the image with the projected pattern, both depth gradient information, and the standard (2D) facial image. Figure 4.8 illustrates some results for subjects in the test set. The first column corresponds to the input to the network, the second column the ground truth texture information, and the

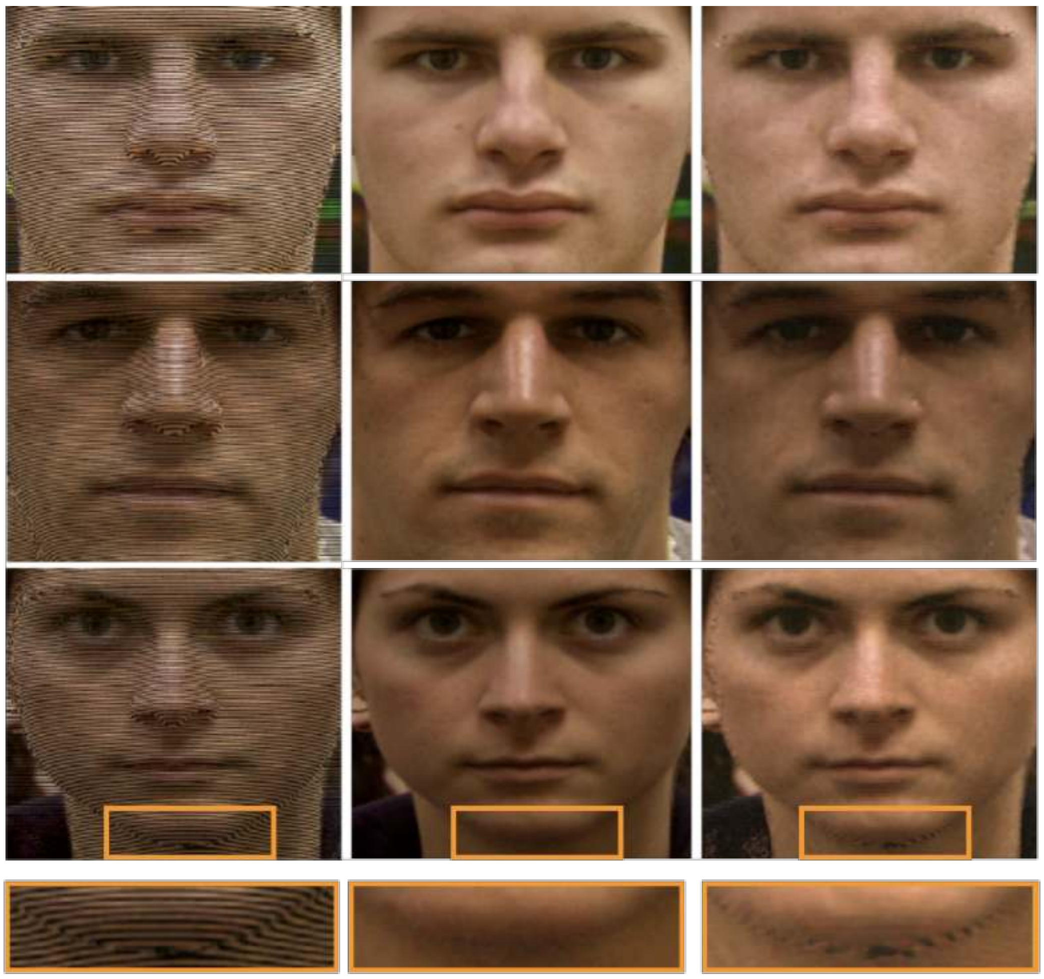


Figure 4.8: Examples of the facial texture recovered from the image with the projected pattern. The first column, shows the input image (denoted as I in Algorithm 1). The second column shows the ground truth, and the third column the texture recovered by the network I_{rgb}. This examples are from the test set and the images associated to these subjects were never seen during the training phase.

third column the retrieved texture information. The architecture of the network and the training protocol is described in detail in Section 4.7. As we can see in the examples illustrated in Figure 4.8, an accurate low resolution texture representation of the face can be achieved in general, and visible artifact are observed only in the regions where the depth is discontinuous (see for example, the regions illustrated at the bottom of Figure 4.8).

Figure 4.9 illustrates the ground truth and the retrieved depth gradient (again, for random samples from the test set). To estimate the 3D information, we feed to a different branch of convolutional layers the gray version of the input image. These layers are fully described in Table 4.5. A gray input image is considered instead of a color one because the projected pattern is achromatic, and therefore, no 3D information is encoded in the colors of the image. In addition, we crop the input image to exclude the edges of the face. (Facial registration and cropping is performed automatically using dlib [43] facial landmarks.) As discussed in Section 4.5, and in particular, in the proof of Proposition 1, the deformation of the projected fringes only provide local gradient information if the norm of the gradient of the depth is bounded. In other words, where the scene present depth discontinuities, no local depth information can be extracted by our proposed approach. This is one of the main reasons why differential 3D information can be exploited for face recognition, while bypassing the more complicated task of a 3D facial reconstruction.

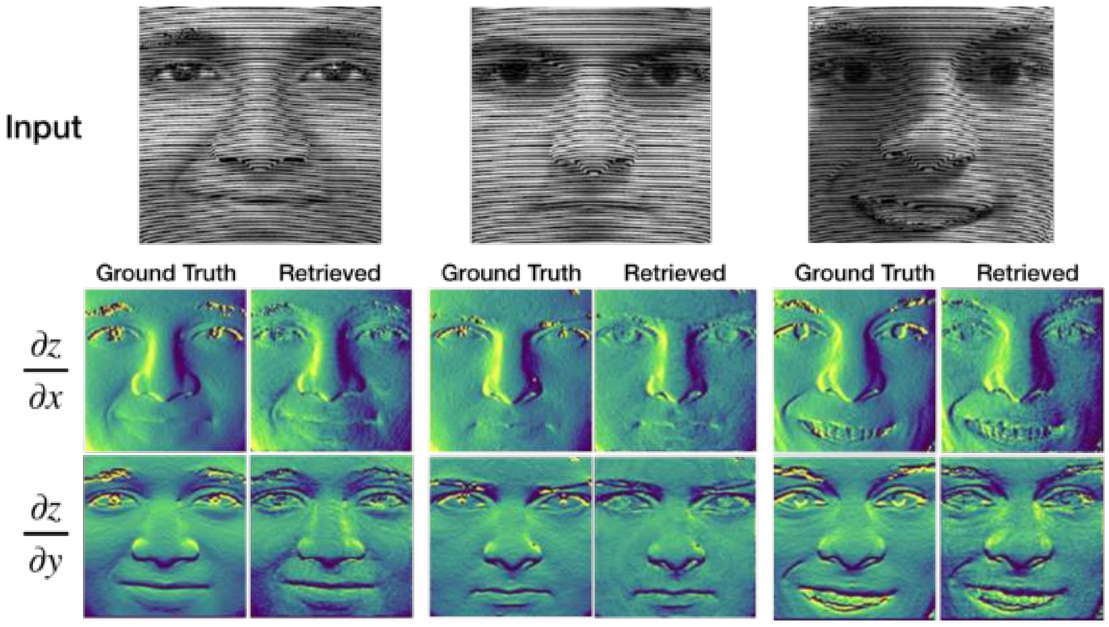


Figure 4.9: Differential depth information extracted from the image with the projected pattern. The first row illustrates the input image (depth information can be extracted from a gray version of the input as the designed patter is achromatic). The second and third row show the ground truth and the retrieved x and y partial derivatives of the depth respectively.

One of the advantages of the proposed approach is that it extracts local depth information, and therefore, the existence of depth discontinuities does not affect the estimation on the smooth portion of the face. This is illustrated in Figure 4.10 (a)-(b), where a larger facial patch is fed into the network. The decomposition module is composed exclusively of convolutional layers, and therefore, images of arbitrary size can be evaluated. Figure 4.10-(a) shows the input to the network, and Figure 4.10-(b) the first channel of the output (for compactness we display only the x-partial derivative). As we can see, the existence of depth discontinuities does not affect the prediction in the interior of the face (we consider the prediction outside this region as noise and we replace it by 0 for visualization).

Several algorithms have been proposed to hallucinate 3D information from a 2D facial image [24, 39, 56, 70]. In order to verify that our decomposition network is extracting real depth information (in lieu of hallucinating it from texture cues), we simulated an image where the pattern is projected over a surface with identical

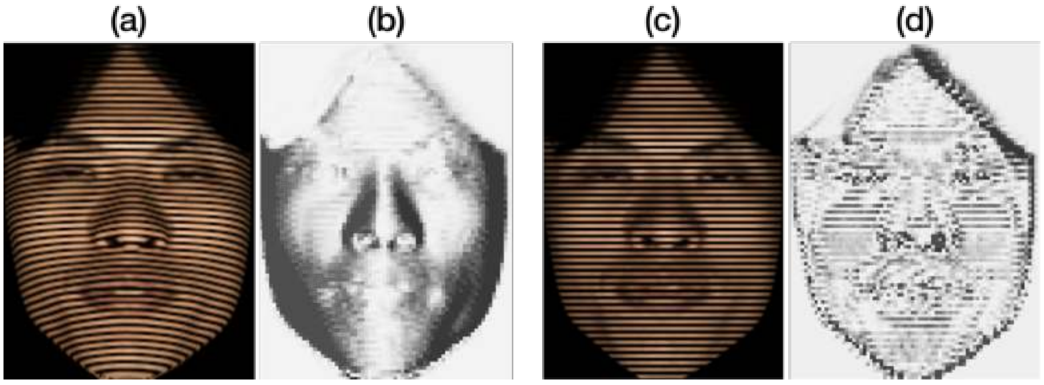


Figure 4.10: Is the network really extracting depth information? In this Figure we show the output of the network for two inputs generated using identical facial texture but different depth ground truth data. (a) Image obtained when the projected pattern is projected over the face with the real texture and the real 3D profile. (b) Output of the network when we input (a) (only the x-partial derivative is displayed for compactness). (c) Image obtained when the projected pattern is projected over a flat surface with the texture of the real face. (d) Output of the network when the input is (c). None of these images were seen during training.

texture but with a planar 3D shape (as in the example illustrated in Figure 4.2). Figure 4.10 (a) shows the image acquired when the fringes are projected over the ground truth facial depth, and (c) when instead the depth is set to 0 (without modifying the texture information). The first component of the output (x-partial derivative) is shown in (b) and (d), as we can see, the network is actually extracting true depth information (from the deformation of the fringes) and not hallucinating 3D information from texture cues. As we will see next, this property is particularly useful for joint face recognition and spoofing prevention.

2D and 3D face recognition. Once the input image is decomposed into a (standard) texture image and depth gradient information, we can proceed to extract 2D and 3D facial features from each component. To this end, state-of-the-art network architectures are evaluated. Our method is agnostic to the RGB and depth feature extractors. Moreover, as the retrieved texture image is close to a standard RGB facial images (in sense of the L2-norm), any pre-train 2D feature extractor can be used (e.g., [14, 69, 77, 90, 117]). In the experiments presented in this Section we tested a network based on the Xception architecture [11] (details are provided in Section 4.7). For the extraction of texture features, the network is trained using FaceScrub [67] dataset (as we previously described, this is a public dataset of 2D facial images). The module that extracts 3D facial features is trained using 2/3 of the subjects of ND-2006 dataset, leaving the remaining subjects exclusively for testing. The output of each module is a 512-dimensional feature vector (see, e.g., Figure 4.4), hence the concatenation of 2D+3D features leads to a 1024-dimensional feature vector. Figure 4.11 illustrates a 2D embedding of the texture features, the depth features, and the combination of both. The 2D mapping is learned by optimizing the t-SNE [60] over the train partition and then a random

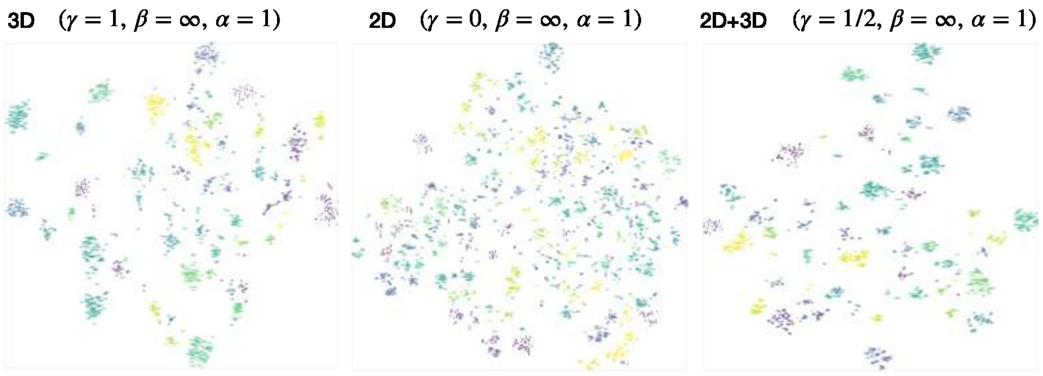


Figure 4.11: Facial features low-dimensional embedding (for visualization purposes only). We illustrate texture-based and depth-based features in a low dimensional embedding space. A random set of subject of the test set is shown. From left to right: the embedding of depth-features, texture-based features, and finally, the combination of texture and depth features. t-SNE [60] algorithm is used for the low-dimensional embedding.

subset of test subjects are mapped for visualization. As we can see, 3D features favor the compactness and increase the distance between clusters associated to different subjects.

To test the recognition performance, the images of the test subjects are partitioned into two sets: gallery and probe. For all the images in both sets, the 2D and 3D feature embedding is computed (using the pre-trained networks described before). Then, for each image in the probe set, the n nearest neighbors in the gallery set are selected. The distance between each sample (in the embedding space) is measured using the distance defined in Section 4.5, Equation (4.12). For each sample in the probe set, we consider the classification as accurate, if at least one of the n nearest neighbors is a sample from the same subject. The rank-n accuracy is the percentage of samples in the probe set accurately classified.

Figure 4.12 and Table 4.1 show the rank-n accuracy when: only 2D features ($\gamma = 0$), only 3D features ($\gamma = 1$), or a combination of both ($0 < \gamma < 1$) is considered. As explained in Section 4.5.3, the value of γ can be used to balance the weight of texture and depth features. As we can see, in all the cases a combination of texture and depth information outperforms each of them individually. This is an expected result as classification tends to improve when independent sources of information are combined [47]. γ is an hyper-parameter that should be set depending on the conditions at deployment. In our particular experiments the best results are obtained for $\gamma = 0{,}3$, which suggests that RGB features are slightly more reliable than depth features. This is an expected result as the module that extract RGB features is typically trained in a much larger datasets (2D facial images became ubiquitous). We believe this may change if, for example, testing is performed under low light conditions [51]. Testing this hypothesis is one of the potential path for future research. In the experiment discussed so far, we ignored the role of β and α (i.e., we set $\beta = \infty$ and $\alpha = 1$). As we will discuss in the following, these parameters become relevant to achieve jointly face recognition

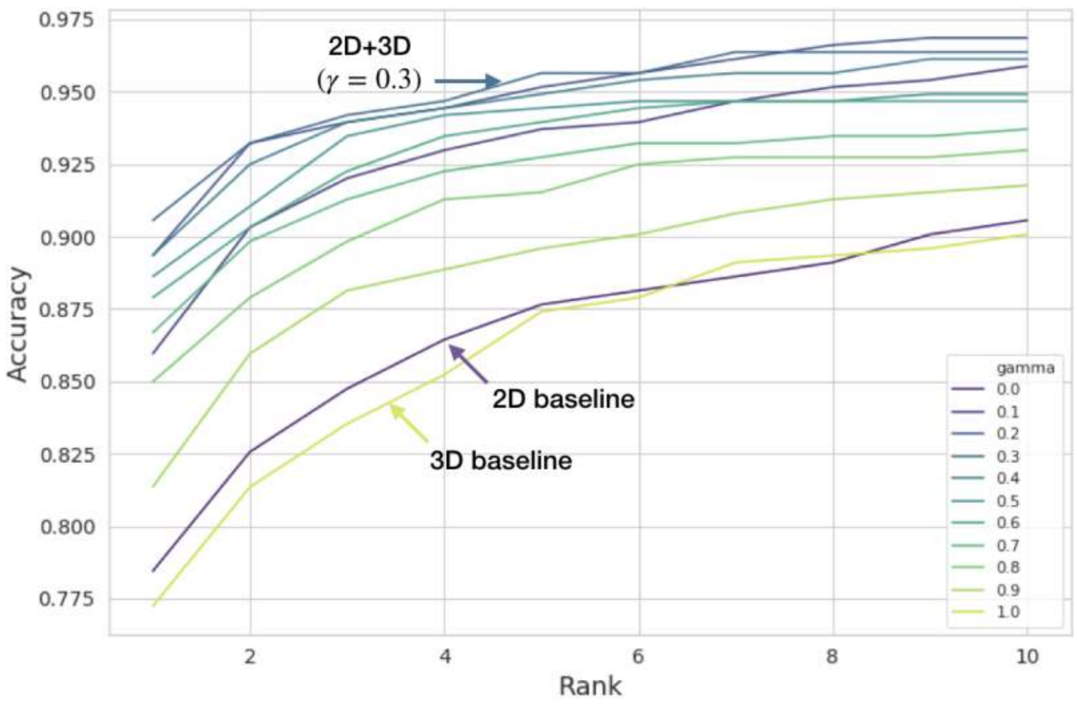


Figure 4.12: Rank-n accuracy for 2D, 3D, and 2D+3D face recognition. As discussed in Section 4.5 the value of γ can be set to weight texture and depth information in the classification decision. The extreme cases are $\gamma = 0$ (only texture is considered) and $\gamma = 1$ (only depth is considered). These extreme cases are illustrated in yellow and purple respectively, while intermediate solutions ($0 < \gamma < 1$) are presented in tones of green.

and spoofing prevention.

Robustness to spoofing attacks. Spoofing attacks are simulated to test face recognition models, and in particular, how robust these frameworks are under (unseen) spoofing attacks. As in the present work we focus on the combination of texture and depth based features, the simulation of spoofing attacks must account for realistic texture and depth models. The models for the synthesis of spoofing attacks are described in detail in Section 4.7.3.

Figure 4.13 illustrates spoofing samples (first four rows) and genuine samples (bottom five rows). The first two columns correspond to the ground truth texture and depth information, the third column illustrates the input to our system, and the last three columns correspond to the outputs of the decomposition network. These three last images are fed into the feature extraction modules for the extraction of texture and depth based features respectively, as illustrated in Figure 4.4. It is extremely important to highlight, that spoofing samples are included exclusively at testing time. In other worlds, during all the training process the entire framework is agnostic to the existence of spoofing examples. If the proposed framework is capable of extracting real 3D facial features, it should be inherently robust to most common types of spoofing attacks.

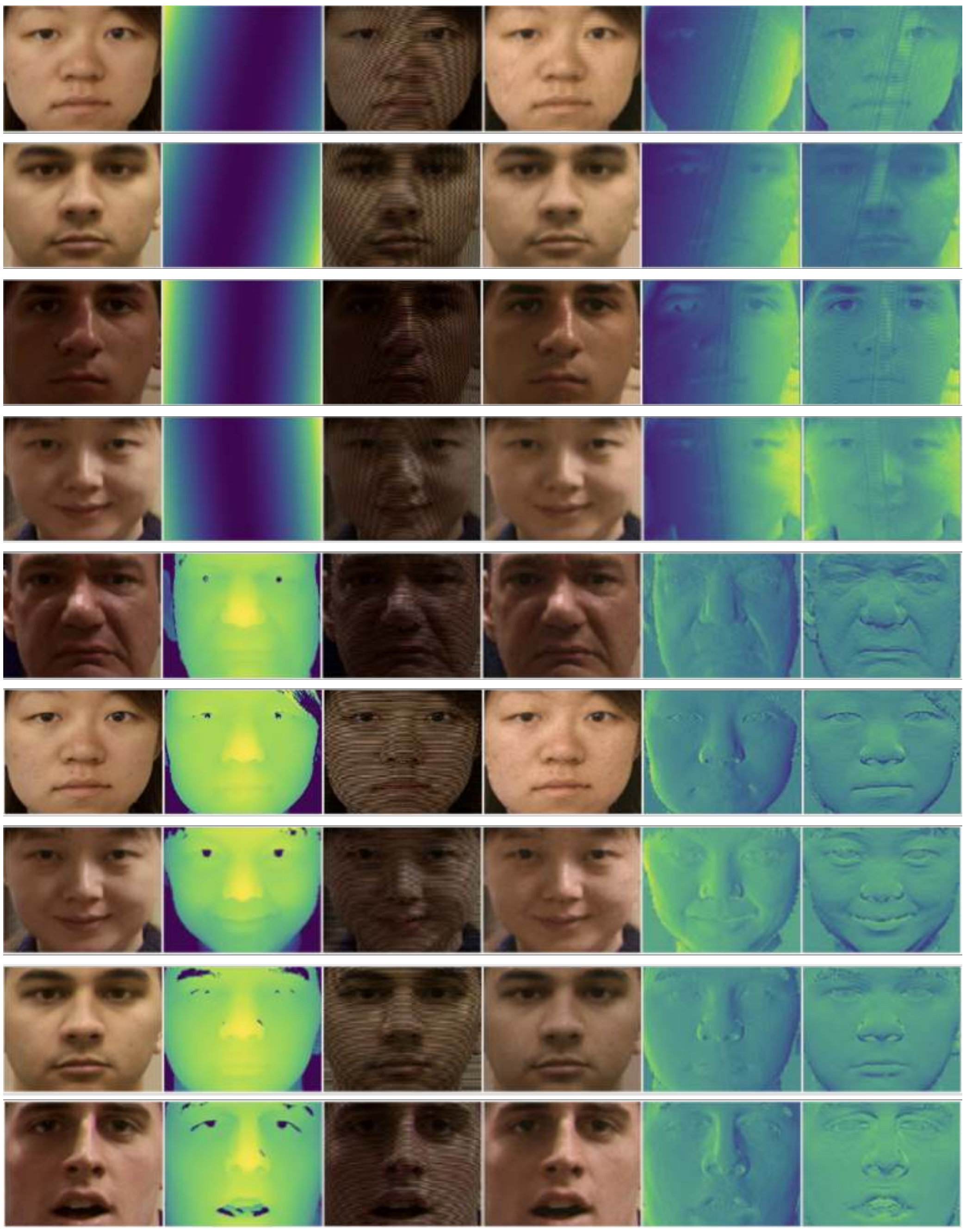


Figure 4.13: Examples of samples from live subjects and spoofing attacks. From left to right: (1) the ground truth texture, (2) the ground truth depth, (3) the input to our system (image with the projected pattern), (4) the recovered texture component (one of the outputs of the decomposition network), (5)/(6) recovered x/y depth partial derivative. The first four rows correspond to spoofing samples (as explained in Section 4.7.3), and the bottom five rows to genuine samples from live subjects.

Table 4.1: Rank-n accuracy for 2D, 3D, and 2D+3D face recognition. As discussed in Section 4.5 the value of γ can be set to weight the impact of texture and depth information. The extreme cases are $\gamma = 0$ (only texture is considered) and $\gamma = 1$ (only depth is considered)

Rank-n Accuracy	1	2	5	10
RGB baseline ($\gamma = 0$)	78,5	82,6	87,7	90,6
Depth baseline ($\gamma = 1$)	77,2	81,4	87,4	90,1
(our) $\gamma = 0{,}3$	**90,6**	**93,2**	**95,6**	**96,4**
(our) $\gamma = 0{,}5$	88,6	91,0	94,4	94,9
(our) $\gamma = 0{,}8$	85,0	87,9	91,5	93,0

As discussed before, the combination of texture and depth based features improves recognition accuracy. On the other hand, when spoofing attacks are included, we observe that texture based features are more vulnerable to spoofing attacks (see for example figure 4.13 and 4.15). To simultaneously exploit the best of each feature component, we design a non-linear distance as described in Equation (4.12). Figure 4.14 illustrates the properties of the defined distance for different values of α and β. As it can be observed, for those genuine samples (relative distances lower than β) the non linear component can be ignored and the distance behave as the euclidean distance with a relative modulation set by γ. On the other hand, if the distance between the depth components is above the threshold β, it will dominate the overall distance achieving a more robust response to spoofing attacks.

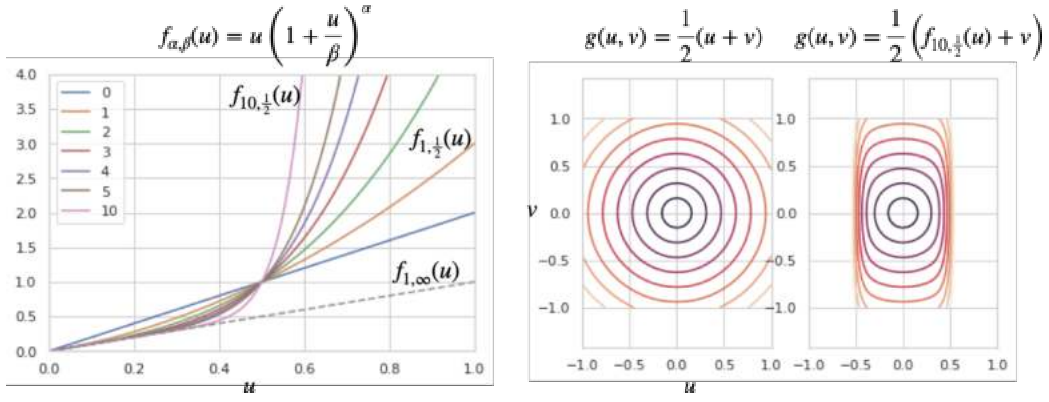


Figure 4.14: Illustration of the properties of the distance function defined in (4.12). On the left side we illustrate the role of the parameter α, and on the right, we compare the proposed distance and the standard euclidean distance. As can be observed, both measures are numerically equivalent in the region $[-\beta/2, \beta/2] \times [-\beta/2, \beta/2]$, but the proposed measure gives a higher penalty to vectors whose u coordinate exceeds the value β.

To quantitatively evaluate the robustness against spoofing attacks, spoofing samples are generated for all the subjects in the test set. As before, the test set is separated into a gallery and a probe set and the generated spoofing samples are aggregated into the probe set. For each image in the probe set, the distance to a sample of the same subject in the gallery set is evaluated. If this distance is below a certain threshold λ, the image is labeled as genuine, otherwise, the image is labeled as spoofing. Comparing the classification label with the ground truth label we obtain the number of true positive (genuine classified as genuine), false positive (spoofing classified as genuine), true negative (spoofing classified as spoofing), and false negative (genuine classified as spoofing). Changing the value of the threshold λ we can control the number of false positive versus the number of false negatives as illustrated in Figure 4.15.

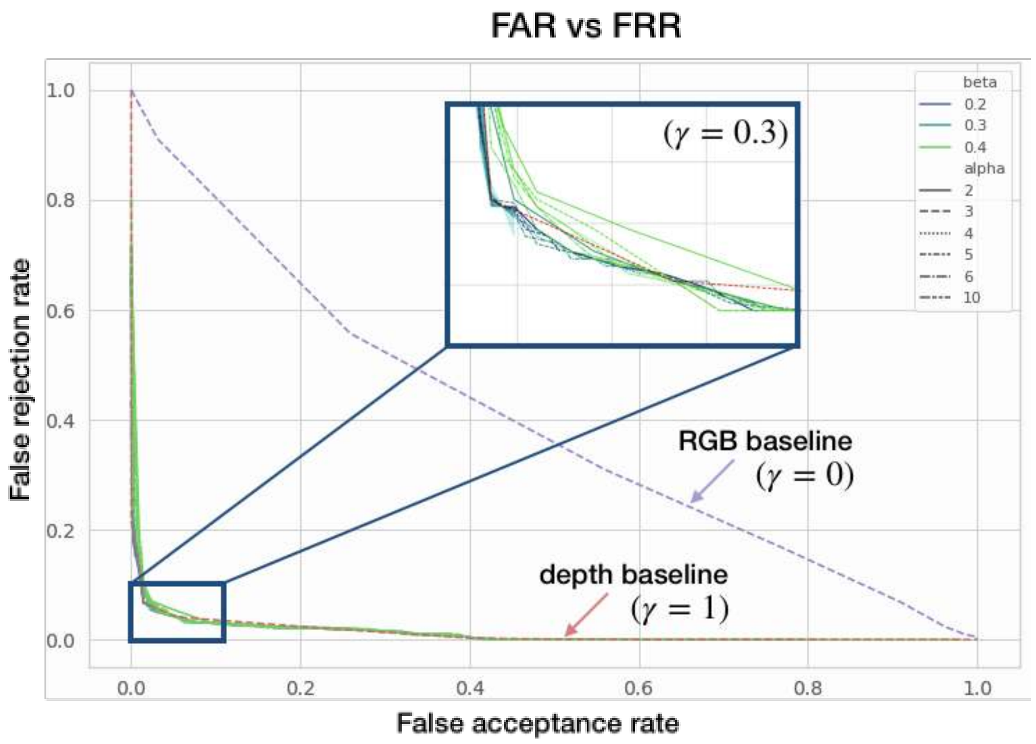


Figure 4.15: False acceptance rate and false rejection rate under the presence of spoofing attacks. On color blue we illustrate the RGB baseline ($\gamma = 0$), on the other extreme, the red curve illustrates the performance when only depth features are considered. The combination of RGB and depth features is illustrated in tones of green for different values of α and β (in this experiment we set $\gamma = 0{,}3$).

Figure 4.15 shows the ratio of false positive and false negative for $\lambda \in [0, 2]$. As before the distance between the samples is computed using the definition provided in (4.12), in blue/red the RGB/depth baseline is illustrated, the other set of curves (displayed in green tones) correspond to a combination of texture and depth features with $\gamma = 0{,}3$ and different values of α and β.

In Table 4.2 the ratio of true positive is reported for a fixed ratio of false positives. The ACER measure (last column) corresponds to the average between the ratio of spoofing and genuine samples misclassified.

Table 4.2: Spoofing detection results. The ratio of true positive for a fixed ratio of false positive and the ACER measure are reported. Texture and depth facial features are combined using the distance defined in (4.12). As we can see, the parameters γ, α, and β can be set to obtain better facial recognition performance and robustness against spoofing detection.

	TPR % @FPR=10^{-3}	TPR % @FPR=10^{-2}	ACER %
RGB baseline ($\gamma = 0$)	21,8	24,0	38,9
Depth baseline ($\gamma = 1$)	**88,4**	**97,1**	4,0
(our) $\gamma = 0{,}3,\ \beta = 0{,}35\ \alpha = 2$	85,5	96,9	4,5
(our) $\gamma = 0{,}3,\ \beta = 0{,}35\ \alpha = 5$	83,8	**97,1**	4,0
(our) $\gamma = 0{,}3,\ \beta = 0{,}35\ \alpha = 10$	85,0	95,6	**3,9**
(our) $\gamma = 0{,}3,\ \beta = 0{,}4\ \alpha = 2$	82,6	96,9	4,7
(our) $\gamma = 0{,}3,\ \beta = 0{,}4\ \alpha = 5$	86,4	**97,1**	4,4
(our) $\gamma = 0{,}3,\ \beta = 0{,}4\ \alpha = 10$	81,8	**97,1**	4,1
(our) $\gamma = 0{,}3,\ \beta = 0{,}5\ \alpha = 2$	86,4	96,4	5,3
(our) $\gamma = 0{,}3,\ \beta = 0{,}5\ \alpha = 5$	82,8	95,6	5,7
(our) $\gamma = 0{,}3,\ \beta = 0{,}5\ \alpha = 10$	85,0	94,4	5,9

Testing variations on the ambient illumination. To test the impact of variations on lighting conditions we simulated test samples under different ambient illumination. Implementation details are described in Section 4.7.4. Table 4.3 compares the rank-5 accuracy of 2D features and 2D+3D features as the power of the ambient illumination increases. The ambient illumination is modeled with random orientation, and therefore, the more powerful the illumination is the more diversity between the test and the gallery samples is introduced.

In the present experiments, we assumed that both the projected pattern and the ambient illumination have similar spectral content. In practice, one can project the pattern, e.g., on the infrared band. This would make the system invisible to the user, and reduce the sensitivity of 3D features to variations on the ambient illuminations. We provide a hardware implementation feasibility study and illustrate how the proposed ideas can be deployed in practice in Section 4.8.

Table 4.3: Recognition accuracy under different ambient illumination conditions. The power of the additional ambient light is provided relative to the power of the projected light, i.e., power=200 % means that the added ambient illumination is twice as bright as the projected pattern.

Rank-5 Accuracy	power=100 %	power=150 %	power=200 %
RGB baseline ($\gamma = 0$)	89,2	81,2	53,9
(our) $\gamma = 0{,}5$	93,6	90,7	80,7

Table 4.4: Spoofing detection results for ArcFace and ArcFace enhanced with 3D features. Like in Table 4.2, the ratio of true positive for a fixed ratio of false positive and the ACER measure are reported.

	TPR % @FPR=10^{-3}	TPR % @FPR=10^{-2}	ACER %
ArcFace ($\gamma = 0$)	0	0	46,2
ArcFace + 3D ($\gamma = 0{,}5$)	84,7	94,7	7,9

Improving state of the art 2D face recognition. To test how the proposed ideas can impact the performance of state-of-the-art 2D face recognition systems, we evaluated our features in combination with texture based features obtained with ArcFace [14]. ArcFace is a powerful method pre-trained on very large datasets, on ND-2006 examples it achieves perfect recognition accuracy (100 % rank-1 accuracy). When ArcFace is combined with the proposed 3D features, the accuracy remains excellent (100 % rank-1 accuracy), i.e., adding the proposed 3D features does not negatively affects robust 2D solutions. On the other hand, 3D features improve ArcFace on challenging conditions as we discuss in the following. Interesting results are observed when ArcFace is tested under spoofing attacks, as we show in Table 4.4, ArcFace fails to detect spoofing attacks. ArcFace becomes more robust when it is combined with 3D features, improving from nearly 0 TPR@FPR(10^{-3}) to 84 %. In summary, as 2D methods improve and become more accurate, our 3D features do not affect them negatively when they work well, while improve their robustness in challenging situations.

4.7. Implementation details

4.7.1. Light projection

From ground truth depth and texture facial information, images under the projection of different high frequency patterns can be simulated as illustrated in

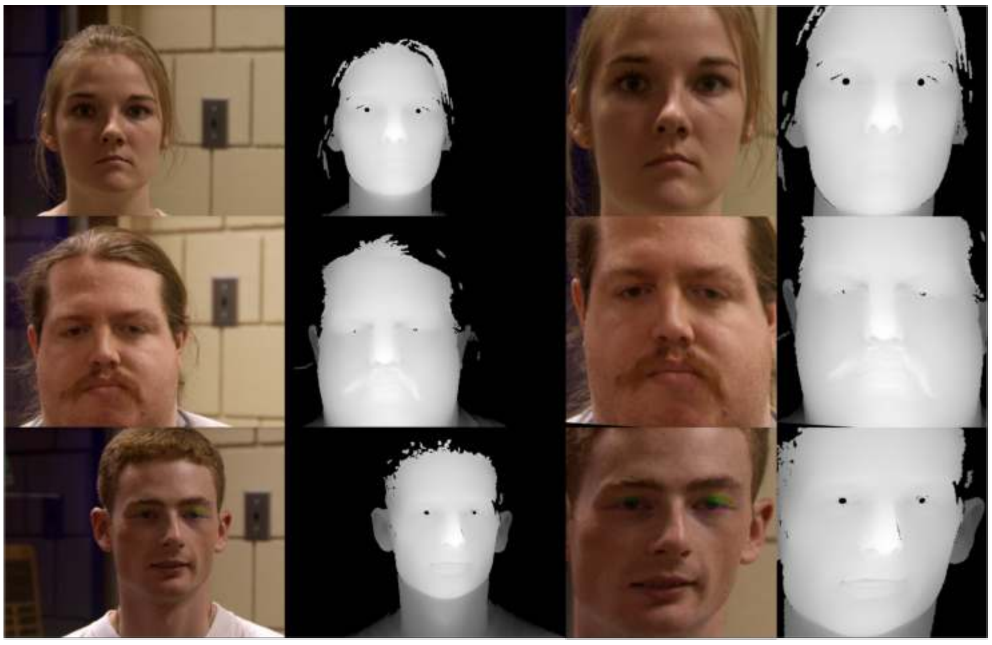


Figure 4.16: Samples from the nd2006 database. On the left column, the RGB images can be seen, while on the right their corresponding depth images are shown.

Figure 4.17. A model of the physical configuration of the system as described in Section 3.2, and different parameters for the baseline and the fringe width were tested. We assumed in all our experiments a fixed focal length. The pseudo-code for the generation of samples under active illumination is summarized in Algorithm 3. It is important to highlight that though the problem of simulating the pattern deformation from the depth is easy, the opposite is a very hard problem. This is why we design a DNN-based approach that estimates from the deformation of the pattern the gradient of the depth rather than the depth itself (see Section 4.5).

4.7.2. Networks architecture

Table 4.5 illustrates the architecture of the network that performs texture and depth information decomposition (as described in Section 4.5). Table 4.6 illustrates the architecture of the layers trained for facial feature extraction (illustrated as yellow/green block in Figure 4.4). Each module of the proposed framework is implemented using standard Tensorflow layers (version 1.13).

Training protocol. The training procedure consists of three phases, first we perform 10 epochs using stochastic gradient descend (SGD), then, we iterate 20 additional epochs using adam optimizer, and finally, we perform 10 epoch using SGD. During these phases the learning rate is set to 10^{-3}. These three steps are commonly refer in the literature as “network warmup”, “training”, and “fine-tuning”. We observed that training each framework module following this protocol leads to

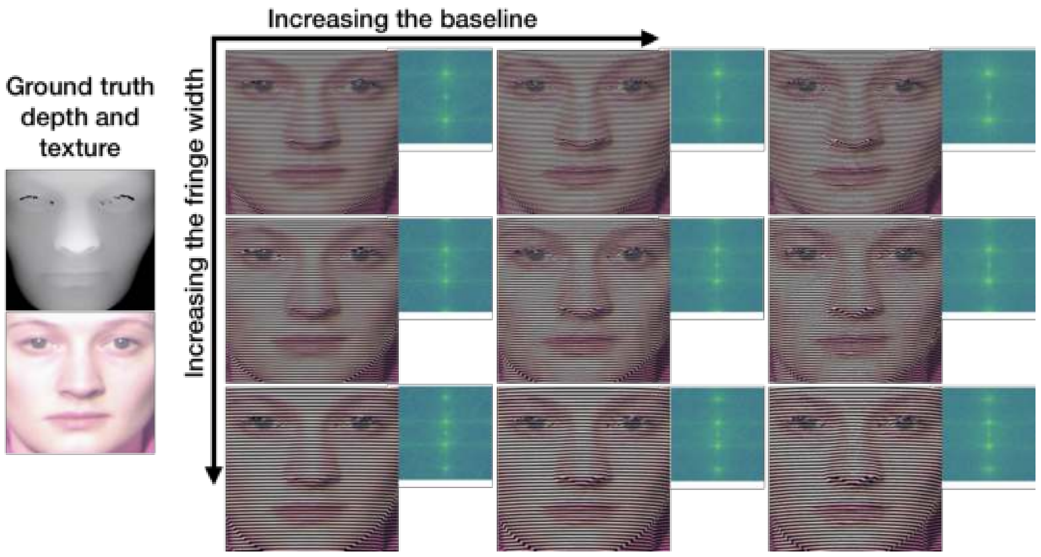


Figure 4.17: Texture and shape information on a single RGB image. On the left side we show the ground truth depth (top) and texture (bottom) facial information for one sample of ND-2006 dataset. With the geometric model described in Section 3.2, we simulated images acquired under the projection of a periodic fringe pattern. The absolute value of the Fourier transform (in logarithmic scale) is illustrated at the right side of each example. From left to right, we show how the baseline (distance between the light source and the camera) impact the simulation. Recall the role of the baseline, defined as b in Equation 3.5. From the top to the bottom, we show the effect of the width of the fringes, which define the fundamental frequency of the pattern $f_0 = 1/T$, see Section 4.5.1. Note that as we are displaying high frequency patterns in the Figure, the reader may be observing aliasing artifacts due to a poor pdf resolution (zooming into the image is recommended).

Algorithm 3 Active light projection. Model the resulting RGB image when a pattern of structured light $p(x, y)$ is projected over a surface with depth profile $D(x, y)$ and texture $T(x, y)$.

```
1: procedure LightProjection($D(x,y), T(x,y)$)
   Read the light pattern to be projected (pre-designed).
2:     $p(x,y) = \text{loadLightProfile}()$
   Simulate the local disparity (see (3.5)).
3:     $disparity(x,y) = \frac{bf}{q^2} D(x,y)$
   Compute local pattern deformation.
4:     $\tilde{p}(x,y) = \text{bilinearInterpolation}(p, (x + d(x,y), y))$
   Account for the texture.
5:     $O(x,y) = \tilde{p}(x,y)\, T(x,y)$
6:     return $O(x,y)$                    ▷ Image with active illumination.
7: end procedure
```

Table 4.5: Decomposition network (illustrated in blue in Figure 4.4). Conv denotes convolution layer, and BN batch normalization. Standard Tensorflow (v1.13) layers are used.

Input:	480×480×3 image with proj. fringes.
(A) RGB - branch	
Layer A.1:	Conv - 64 kernels 4×4 , BN, LeakyRelu.
Layer A.2:	Conv - 4 kernels 3×3 , BN, LeakyRelu.
Layer A.3:	Conv - 4 kernels 3×3 , BN, LeakyRelu.
Layer A.4:	Conv - 2 kernels 1×1 , BN, LeakyRelu.
Layer A.5:	Resize(112×112×3)
Output A:	112×112×3 recovered texture.
(B) Depth - branch	
Layer B.1:	Average(axis=3) (convert to gray).
Layer B.2:	Conv - 64 kernels 4×4 , BN, LeakyRelu.
Layer B.3:	Conv - 4 kernels 3×3 , BN, LeakyRelu.
Layer B.4:	Conv - 4 kernels 3×3 , BN, LeakyRelu.
Layer B.5:	Conv - 2 kernels 1×1 , BN, LeakyRelu.
Layer B.6:	Resize(112×112×2)
Output B:	112×112×2 recovered $\{z_x, z_y\}$.

stable and satisfactory results (as reported in Section 4.6). However, we did not focus in the present work on the optimization of the networks architecture, nor the training protocols.

4.7.3. Simulation of spoofing attacks

Spoofing attacks are simulated to test face recognition models, in particular, how robust these frameworks are under (unseen) spoofing attacks. As in the present work we focus on the combination of texture and depth based features, the simulation of spoofing attacks must account for realistic texture and depth models. Algorithm 4 summarizes the main steps for the simulation of spoofing attacks (which are detailed next).

To simulate realistic texture conditions, CASIA dataset is analyzed [111]. This dataset contains samples of videos collected under diverse spoofing attacks, e.g., video and photo attacks. 50 different subjects participated in the data collection, and 600 video clips were collected. We extracted a subset of random frames from these videos (examples are illustrated in Figure 4.18), and use them to identify certain texture properties that characterize spoofing attacks, as we explain next.

Several works have been published in the recent years supporting that certain texture patterns are characteristics of spoofing photographs [3, 53–55, 58, 102]. In particular, differences in the Fourier domain have been reported [54], which provide cues for certain classes of spoofing attacks [102]. Following these ideas, we propose a simple model for the synthesis of spoofing attacks from genuine samples extracted

Algorithm 4 Synthesis of the texture and depth of spoofing attacks.

```
 1: procedure TransformText($T_0$)                              ▷ Sim. spoofing texture
    Init. optimal filter (this is done only once and off-line).
 2:     $k(x, y)$ = InitKernel(RealEx., SpoofEx.)                 ▷ (4.13)
    Filter the genuine sample.
 3:     $T(x, y)$ = filter($T_0, k(x, y)$)
 4:     return $T(x, y)$                                        ▷ Simulated spoofing texture
 5: end procedure
 6: procedure TransformDepth                                    ▷ Sim. spoofing depth
    Compute spoofing depth
 7:     $z(x, y)$ = SpoofDepthModel()                             ▷ E.g., (4.15),(4.16).
 8:     return $z(x, y)$                                        ▷ Simulated spoofing depth
 9: end procedure
10: procedure SpoofingSample($\{T_0, z_0\}$)
    Simulate sample texture.
11:     $T(x, y)$ = TransformText($T_0$)
    Simulate sample depth.
12:     $z(x, y)$ = TrasformDepth()
13:     return $\{T, z\}$
14: end procedure
```

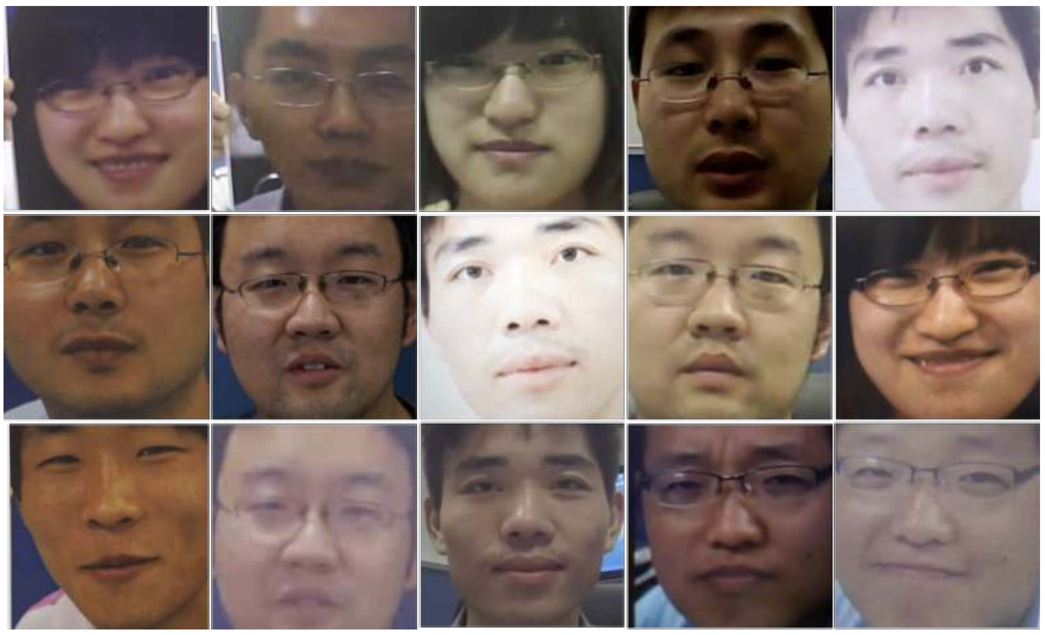


Figure 4.18: Spoofing and genuine examples from CASIA [111] dataset.

Table 4.6: Feature embedding (illustrated in yellow/green in Figure 4.4). Conv denotes convolution layer, SepConv separable convolution, BN batch normalization, IN instance normalization. Standard tensorflow (v1.13) layers are used.

Input:	112×112×c ($c = 3$ for texture, $c = 2$ for depth image).
Layer 1.1:	Conv - 32 kernels 3×3 stride 2, IN, Relu.
Layer 1.2:	Conv - 64 kernels 3×3 stride 2, IN, Relu.
Path A1	
Layer A1.1:	Conv - 128 kernels 1×1 stride 2, IN.
Path B1	
Layer B1.1:	SepConv - 128 kernels 3×3, IN, Relu.
Layer B1.2:	SepConv - 128 kernels 3×3, IN, Relu.
Layer B1.3:	MaxPooling 2×2.
Layer 2:	Path A1 + Path B1 (output B1.3 + output A1.1)
Path A2	
Layer A2.1:	Conv - 384 kernels 1×1 stride 2, IN.
Path B2	
Layer B2.1:	SepConv - 384 kernels 3×3, IN, Relu.
Layer B2.2:	SepConv - 384 kernels 3×3, IN, Relu.
Layer B2.3:	MaxPooling 2×2.
Layer 3:	x = Path A2 + Path B2 (output B2.3 + output A2.1)
Middle flow:	x: 28 × 28 × 384 (Repeat 2 times)
Layer 4.1:	SepConv - 384 kernels 3×3, IN, Relu.
Layer 4.2:	SepConv - 384 kernels 3×3, IN, Relu.
Layer 4.3:	SepConv - 384 kernels 3×3, IN, Relu.
Layer 4.4:	Add(x, output Layer 4,3)
Exit flow:	28 × 28 × 256
Path A5	
Layer A5.1:	Conv - 512 kernels 1×1 stride 2, IN.
Path B5	
Layer B5.1:	SepConv - 256 kernels 3×3, IN, Relu.
Layer B5.2:	SepConv - 512 kernels 3×3, IN, Relu.
Layer B5.3:	MaxPooling 2×2.
Layer 6:	Path A5 + Path B5 (output B5.3 + output A5.1)
Layer 7:	SepConv - 512 kernels 3×3, IN, Relu.
Layer 8:	SepConv - 512 kernels 3×3, IN, Relu.
Layer 9:	Global Average Pooling 2D.
Output:	512 × 1 facial features.

from ND-2006 dataset. We assume a generic linear model $I_{spoof}(x, y) = I_{real}(x, y) * k(x, y)$, where I_{spoof} represents the texture of the simulates spoofing attack for the genuine sample I_{real} and k an arbitrary kernel that will be set.

Let us denote as $g_i(x,y)$ $(i = 1,...,M)$ a set of facial images associated to spoofing attacks (here extracted from CASIA dataset), and $q_i(x,y)$ $(i = 1,...,N)$ a set of genuine facial images (for example, from CASIA or ND-2006 datasets). As in Section 4.5 we denote $\tilde{q}(f_x, f_y) = \mathcal{F}\{q(x,y)\}$ the 2D discrete Fourier transform of $q(x,y)$. Given ground truth examples of genuine and spoofing facial images (q_i and g_i), we define the kernel k as

$$k(x,y) = \mathcal{F}^{-1}\left\{\frac{Q(f_x,f_y)}{G(f_x,f_y)+\epsilon}\right\}, \text{ s.t. } k(-x,-y) = k(x,y), \tag{4.13}$$

where G and Q are defined as follow

$$G = \frac{1}{M}\sum_{i=1}^{M} \|\tilde{g}_i(f_x,f_y)\|, \; Q = \frac{1}{N}\sum_{i=1}^{N} \|\tilde{q}_i(f_x,f_y)\|. \tag{4.14}$$

We set $\epsilon = 10^{-6}$ for numerical stability. Observe that G and Q account only for the absolute value of the Fourier transform of real and spoofing samples, while the phase information is discarded. In principle, an arbitrary phase factor can be included. We constrain the solution to be a symmetric kernels, which is equivalent to enforce a null phase. To perform the average defined in Equation (4.14), the 2D coordinates associated to the frequency domain must be refer to a common coordinate frame, this allows to aggregate the frequency information of images of heterogeneous resolution.

Figure 4.19-(a) shows the kernel obtained using 600 spoofing samples from CASIA dataset and 600 genuine samples from ND-2006. As we can see, the resulting kernel is composed of predominantly positive values, this suggest that the application of it will produce essentially a blurred version of the original image. This empirical result is in accordance with the findings reported in [54, 102]. Figure 4.19-(b) shows an example of an image of a genuine face (left), the result of applying the estimated kernel k (center) and the different between them (right).

Finally, the depth profile of different types of spoofing attacks is simulated. To this end, different kinds of polynomial forms were evaluated. We simulated planar attacks, which take place when the attack is deployed using a phone or a tablet, and non-planar attacks, common when the attacker uses a curved printed portrait. We modeled the latter as random parabolic 3D shapes. The principal axis of each surface is randomly oriented, presenting an angle $\theta \sim \mathcal{N}(0, (1/10 \cdot \pi/2)^2)$ with respect to the vertical (as illustrates Figure 4.20). The depth profile z(x,y) is given by

$$z(x,y) = \frac{4a}{w^2}\left(u(x,y) - c\frac{w}{2}\right)^2 \tag{4.15}$$

where $u(x,y) = x\cos(\theta) + y\sin(\theta)$, w is a constant representing the width of the spatial domain (here 480 pixels), a sets the curvature of and is a random variable sampled from the normal distribution $\mathcal{N}(0, (2/10)^2)$, and c (also a random variable) sets the offset of the principal axis and is sampled from the distribution $\mathcal{N}(0, (1/10)^2)$. We also tested arbitrary polynomial forms with random coefficients,

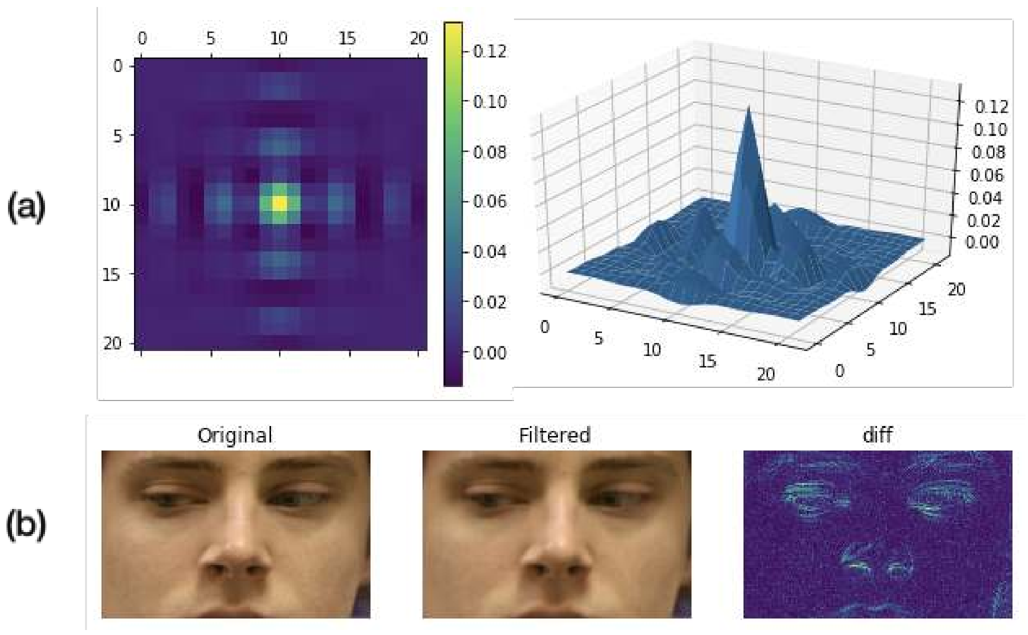


Figure 4.19: Texture simulation of spoofing attacks. (a) Illustrates the linear kernel optimized such that the Fourier domain of real samples match that of spoofing samples. (b) Shows (left) an example of an image of a genuine subject (from ND-2006 set), the simulated texture of a spoofing attack (center), and finally (right) the different between the two.

e.g.,

$$z(x,y) = \sum_{\substack{i=0,..,3\\ j=0,..,3}} w_{ij}\, a_{ij}(x/w - b_{ij})^i (y/h - c_{ij})^j \tag{4.16}$$

where the coefficients are samples from an uniform distribution $a_{ij}, b_{ij}, c_{ij} \sim \mathcal{U}_{[0,1]}$, and $w_{ij} = 10^{\text{máx}\,(0,i-1)\,\text{máx}\,(0,j-1)}$ is a normalization factor.

Figure 4.20 illustrates (right side) some examples of spoofing depth profiles generated.

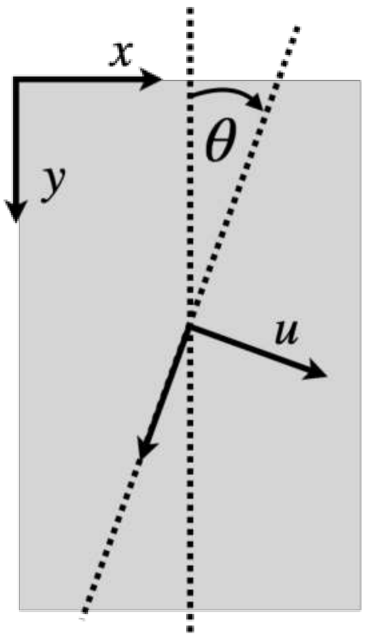
4.7.4. Facial appearance under different illumination

Using the original texture and depth facial information, the appearance of the face under novel illumination conditions can be simulated. One of the simplest and more accepted models consists of considering the facial surface as a lambertinan surface [5, 79, 115]. Hence, the amount of light reflected can be estimated as proportional to the cosine of the angle between the surface local normal and the direction in which the light approaches the surface.

More precisely,

$$I_{(x,y,z)} = a_{(x,y,z)}\ \text{máx}\left(\vec{n}_{(x,y,z)} \cdot p_0\vec{l},\ 0\right), \tag{4.17}$$

where I denotes the intensity of the light reflected, p_0 the intensity of the incident



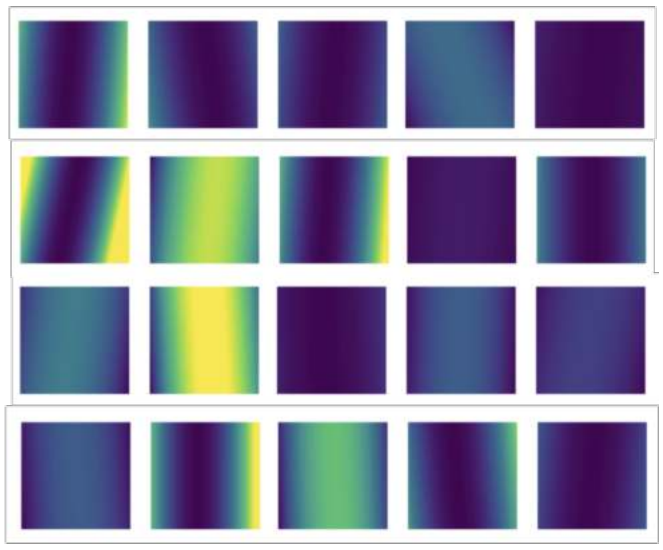


Figure 4.20: Depth simulation of the spoofing attacks. On the left we illustrate the geometry of the simulation process. The right side illustrates examples of the generated depth profiles associates to spoofing attacks.

light, a represents the surface albedo, $\vec{n}$ is a unit vector normal to the surface, and $\vec{l}$ a unit vector indicating the direction in which the light rays approach the surface at $(x, y, z(x, y))$.

If the depth profile $z(x, y)$ is know, the normal vector to the surface at (x, y, z) can be computed as

$$\vec{n}(x,y) = \frac{1}{\sqrt{\left(\frac{\partial z}{\partial x}\right)^2 + \left(\frac{\partial z}{\partial y}\right)^2 + 1}} \left(-\frac{\partial z}{\partial x}, -\frac{\partial z}{\partial y}, 1\right)^t . \tag{4.18}$$

Algorithm 5 summarizes the main steps for simulating new samples under novel illumination conditions. Figure 4.21 shows some illustrative results for illuminants of different intensity and located at different relatives positions with respect to the face.

4.8. Hardware implementation: a feasibility study

We tested a potential hardware implementation of the proposed ideas using commercially available hardware. To project the fringe pattern we used a projector EPSON 3LCD with native resolution 1920 × 1080, and a webcam Logitech C615 with native resolution 1280 × 720. The features of this particular hardware setup are independent of the proposed ideas. One could choose, for example, projecting infrared light, or design a specific setup to meet specific deployment conditions.

Algorithm 5 Steps for the simulation of additional ambient light. The inputs represent: I_0 the facial albedo, z the facial depth map, p_0 the intensity of the source light, l the direction in which the light source is located. Examples are shown in Figure 4.21.

```
 1: procedure addAmbientLight($I_0, z, p_0, l$)
    Make sure the depth map is smooth and noise-free.
 2:     $z$ = Denoising($z$)            ▷ (Gradient op. amplifies noise.)
    Compute depth gradient.
 3:     $z_x, z_y$ = ComputeGradient($z$)
    Compute surface normals (4.18).
 4:     $n = [-z_x, -z_y, 1]$
 5:     $n = \frac{n}{\|n\|_2}$
    Additional light factor (due to the new source).
 6:     $I_+[x, y] = p_0 \text{ máx}\,(n(x, y) \cdot l,\ 0)$
    For each color add the additional brightness.
 7:     $I[x, y, c] = I_0[x, y, c](1 + I_+[x, y])$
    Model camera saturation.
 8:     $I$ =clip($I$, $[0, 1]$)
 9:     return $I$
10: end procedure
```

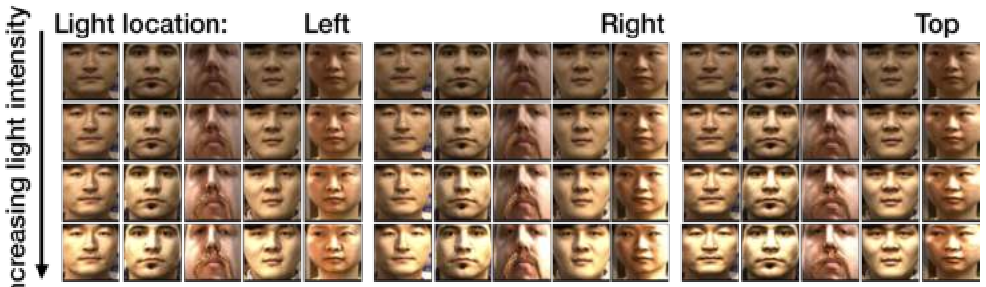


Figure 4.21: Examples of new samples under different lighting conditions. The groups on the left, middle and right, correspond to a light source located at the left, right, and top of the scene respectively. The images on the bottom are created assuming a brighter light source (higher value of p_0, see Algorithm 5).

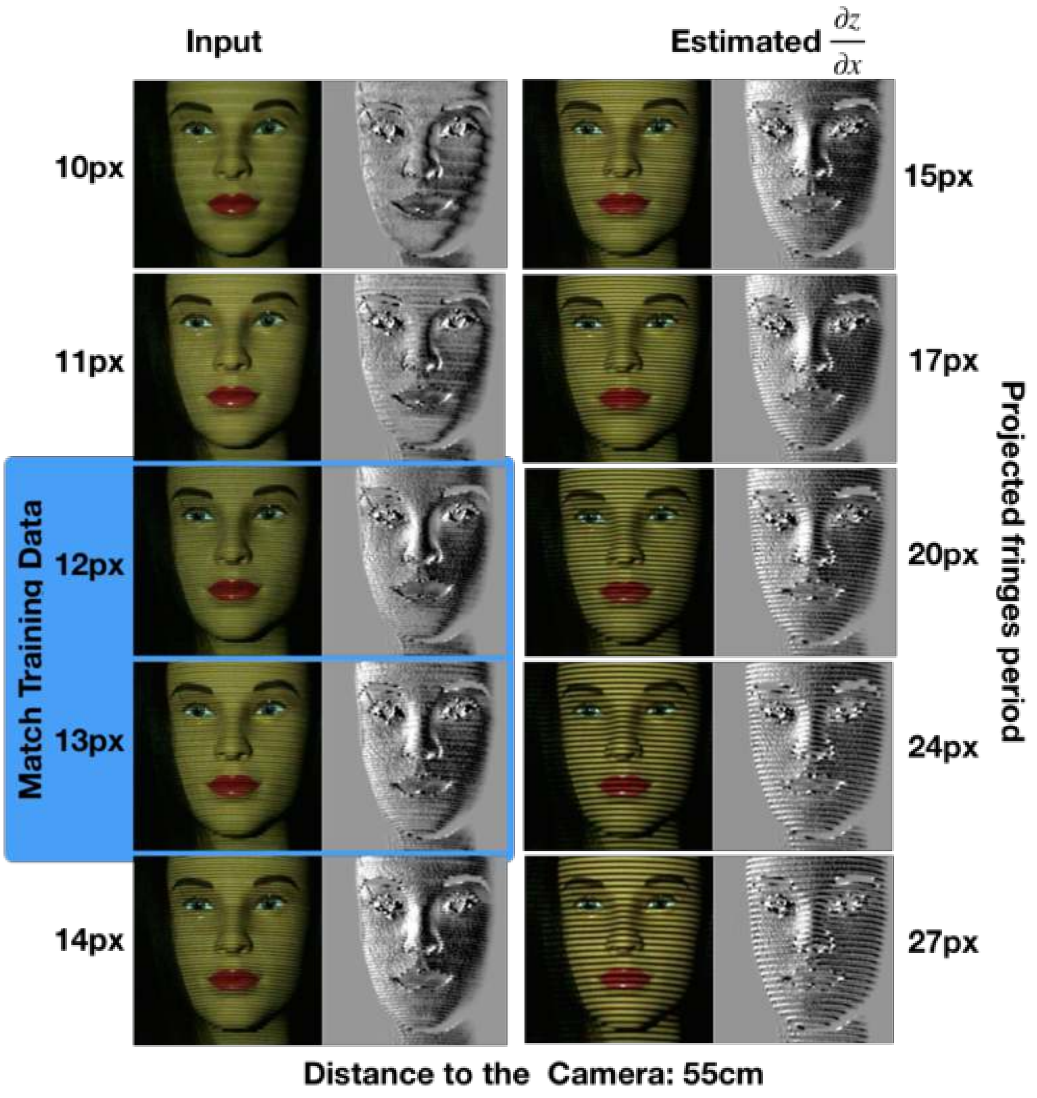


Figure 4.22: Testing the model on a hardware implementation. We projected the proposed light pattern using a commercial projector (EPSON 3LCD) and captured facial images using a standard webcam (Logitech C615). Side by side, we show the captured image (left) and the x-partial derivative of the depth estimated by our DNN model. We tested projected patters of different period, ranging from 10px to 27px (pixels measured in the projector sensor).

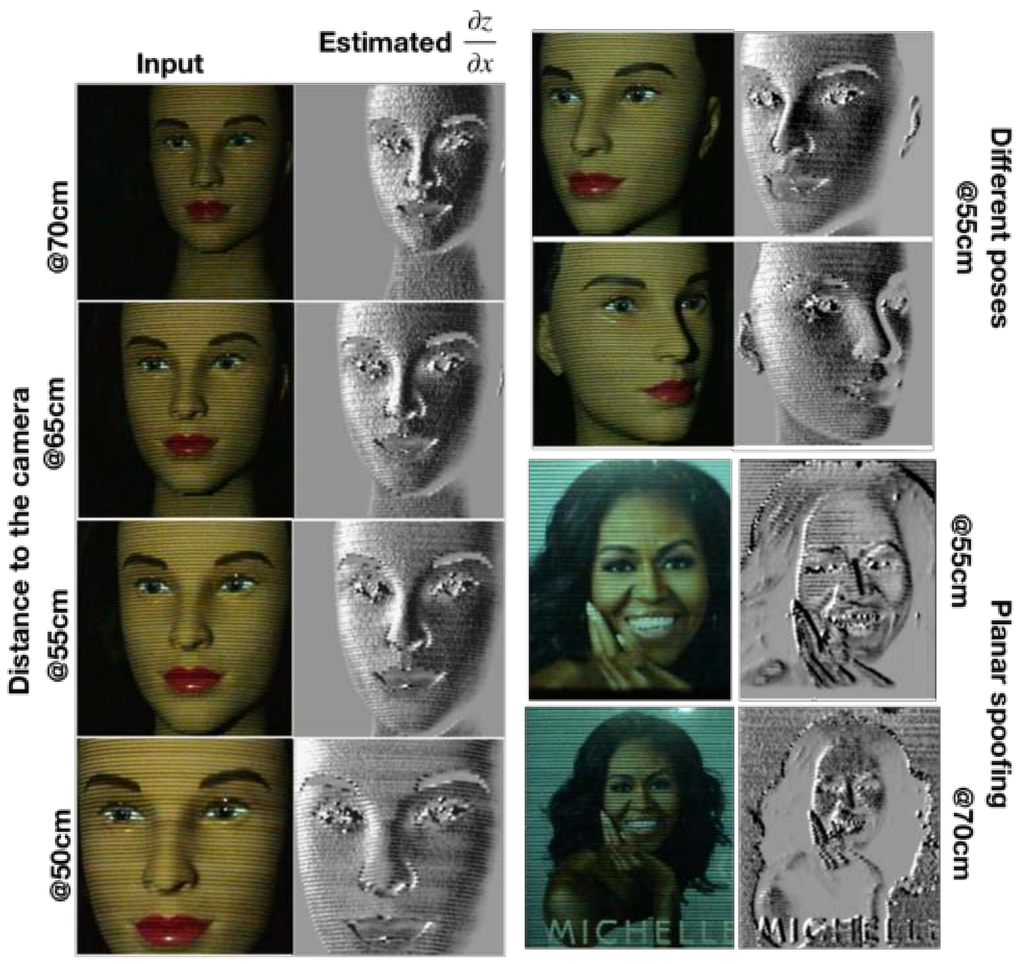


Figure 4.23: Changing the distance to the camera and pose. Complementing the results shown in Figure 4.22, we tested how our DNN model perform as the distance to the camera-project system changes. The left column shows side by side the image captured by the camera and the partial derivative of the depth estimated by our pre-trained model. The distance of the test face to the camera ranged from 50cm to 70cm. The left column shows on the top the results obtained for different head-poses, and on the bottom, the output when a planar facial portrait is presented.

Figure 4.22 shows some empirical results obtained by photographing a mannequin at different relative positions and under different illumination. We qualitative tested how the width of the projected pattern affects the estimation performance. In addition, we tested (see Figure 4.23) how the distance of the face to the camera-projector system affects the prediction of the depth derivative. These two experiments are related as changing the distance to the camera changes the camera perception of the fringes width.

It is important to take into account that in this experiments we tested our pre-trained models (completely agnostic to this particular hardware implementation), i.e., no fine-tuning or calibration was performed prior capturing the images presented in Figures 4.22 and 4.23. In production settings, in contrast, one would fix a specific hardware setup, collect new ground truth data, and fine-tune the models to optimize the setup at hand.

4.9. Conclusions

We proposed an effective and modular alternative to enhance 2D face recognition methods with actual 3D information. A high frequency pattern is designed to exploit the high resolution cameras ubiquitous in modern smartphones and personal devices. Depth gradient information is coded in the high frequency spectrum of the captured image while a standard texture facial image can be recovered to exploit state-of-the-art 2D face recognition methods. We show that the proposed method can be used to leverage 3D information and texture information simultaneously. This allows us to enhance state-of-the-art 2D methods improving their accuracy and making them robust, e.g., to spoofing attack.

Chapter 5

Conclusions and Future Work

Machine Learning and Deep Learning have had an enormous impact in a myriad of fields. In Face Recognition, these techniques have speed up the development of the area, achieving incredible results.

In this work we did a brief introduction to the concepts involved in Machine Learning. We reviewed the ideas and the inner works in Deep Learning, keeping in mind how it developed throughout the years. Then, we focused on Face Recognition, presenting the problem and diving into the different groundbreaking methods that relied on neural networks proposed in the last few years. Finally, we focused on our proposed novel method to perform Face Recognition with implicit 3D cues. That is, without depth reconstruction.

Deep Learning techniques unlock a number of applications and possibilities, and its proper use can help us to improve different processes. Studying the proposed methods, understanding the different techniques involved and proposing new techniques is important both to keep advancing the knowledge in the field, and also being aware of what can be done with them. There are a number of reasons why we need to be as efficient with our resources and processes as possible, and the advancements made with deep learning are a step in that direction. At the same time, there are some ethical and moral discussions to be had around these techniques (which is usual with every technological and scientific advancement), and this is not possible without a proper understanding of their inner workings and capabilities.

In the context of this thesis, we proposed, presented internationally, and published a novel face recognition technique. Personally, I was involved practical matters surrounding this article: code and perform experiments, graphs, build the different networks used, and so on. Although I was part of theoretical discussions as well, the ideas and theory behind this paper were proposed by my advisors and coauthors, who are experts on this field. I was lucky enough to deepen my knowledge in this area during this process by working closely with them.

In the future, it would be interesting to develop the proposed technique further. In the presented work, it can be seen that most immediate benefits are seen in preventing spoofing attacks. While it is proven that the performance accuracy is not hurt by adding this network on top of another state of the art 2D recognition network, it would be interesting to try to find conditions in which identification is improved. By having more representational power due to consider 3D information, it is possible to imagine scenarios in which accuracy is also improved at the same time that spoofing attacks are prevented.

Another possible area where this method can be enhanced is to try to optimize it when deploying it in a real hardware configuration. It would be interesting to explore and solve the challenges associated with a low-cost hardware implementation in a real scenario (e.g., interaction with ambient light). Facing them and try to come up with solutions for them might improve the technique as a whole, understanding better its limitations.

Finally, the training of the network can be studied more in depth and more expermentation can be made. The code can be improved in order to be more plug-and-play with different networks and frameworks. Two possible paths that may be pursued are: gathering a larger dataset, with more variability, to improve the robustness of the depth recovery network, and keep studying different networks configurations of the network as a whole. The module that is tasked with separating 3D information from 2D can be also improved. In this work it is not trained with spoofing samples. By feeding spoof samples, it might improve its performance when recovering shapes that are planar, which can also translate to better separability between real and spoof samples.

There are definitely a lot of areas in this work that can be studied more in depth: Applying interpretability tools to dissect what DNNs are learning. Leveraging the power of novel and larger facial databases. Substituting the pre-defined designed distance with a different one completely learnt from real and spoofing samples. Performing ablation studies with different architectures to better identify the contribution to facial identification from 3D and 2D components. It is my hope that work on these areas is continued so a better understanding of this method potential and drawbacks can be achieved.

Biblography

[1] Amit Agrawal, Rama Chellappa, and Ramesh Raskar. An algebraic approach to surface reconstruction from gradient fields. In *Tenth IEEE International Conference on Computer Vision (ICCV'05)*, volume 1, pages 174–181. IEEE, 2005.

[2] Amit Agrawal, Ramesh Raskar, and Rama Chellappa. What is the range of surface reconstructions from a gradient field? In *Computer Vision – ECCV 2006*, pages 578–591. Springer, 2006.

[3] Yousef Atoum, Yaojie Liu, Amin Jourabloo, and Xiaoming Liu. Face anti-spoofing using patch and depth-based CNNs. *International Joint Conference on Biometrics (IJCB), IEEE*, pages 319–328, 2017.

[4] Gastón A Ayubi, Jaime A Ayubi, J Matías Di Martino, and José A Ferrari. Pulse-width modulation in defocused three-dimensional fringe projection. *Optics Letters*, 35(21):3682–3684, 2010.

[5] Ronen Basri and David W Jacobs. Lambertian reflectance and linear subspaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(2):218–233, 2003.

[6] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. *CoRR*, abs/1212.0901, 2012.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, New York, 2006.

[8] Volker Blanz and Thomas Vetter. Face recognition based on fitting a 3d morphable model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1063–1074, 2003.

[9] Zinelabinde Boulkenafet, Jukka Komulainen, Lei Li, Xiaoyi Feng, and Abdenour Hadid. OULU-NPU: A mobile face presentation attack database with real-world variations. *IEEE International Conference on Automatic Face and Gesture Recognition*, May 2017.

[10] Kaidi Cao, Yu Rong, Cheng Li, Xiaoou Tang, and Chen Change Loy. Pose-robust face recognition via deep residual equivariant mapping. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5187–5196, 2018.

[11] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258, 2017.

[12] Jiyun Cui, Hao Zhang, Hu Han, Shiguang Shan, Xilin Chen, and Intelligence Technology. Improving 2d face recognition via discriminative face depth estimation. pages 140–147, 2018.

[13] Jiankang Deng, Shiyang Cheng, Niannan Xue, Yuxiang Zhou, and Stefanos Zafeiriou. UV-GAN: Adversarial facial UV map completion for pose-invariant face recognition. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7093–7102, 2018.

[14] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou. Arcface: Additive angular margin loss for deep face recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4690–4699, 2019.

[15] Jiankang Deng, Jia Guo, Yuxiang Zhou, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-stage dense face localisation in the wild. *CoRR*, abs/1905.00641, 2019.

[16] J Matías Di Martino, Alicia Fernández, and José A Ferrari. One-shot 3d gradient field scanning. *Optics and Lasers in Engineering*, 72:26–38, 2015.

[17] J. Matías Di Martino, Fernando Suzacq, Mauricio Delbracio, Qiang Qiu, and Guillermo Sapiro. Differential 3d facial recognition: Adding 3d to your state-of-the-art 2d method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(7):1582–1593, 2020.

[18] Matías Di Martino, Jorge Flores, and José A Ferrari. One-shot 3d scanning by combining sparse landmarks with dense gradient information. *Optics and Lasers in Engineering*, 105:188–197, 2018.

[19] Changxing Ding and Dacheng Tao. A comprehensive survey on pose-invariant face recognition. *ACM Transactions on Intelligent Systems and Technology*, 7(3), 2016.

[20] Pengfei Dou, Shishir K Shah, and Ioannis A Kakadiaris. End-to-end 3D face reconstruction with deep neural networks. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5908–5917, 2017.

[21] Zhouyu Du, Antonio Robles-Kelly, and Fangfang Lu. Robust surface reconstruction from gradient field using the l1 norm. In *9th Biennial Conference of the Australian Pattern Recognition Society on Digital Image Computing Techniques and Applications (DICTA 2007)*, pages 203–209. IEEE, 2007.

[22] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[23] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2650–2658, 2015.

[24] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems*, volume 27, pages 2366–2374, 2014.

[25] Timothy C. Faltemier, Kevin W. Bowyer, and Patrick J. Flynn. Using a multi-instance enrollment representation to improve 3d face recognition. In *Proc. First IEEE International Conference on Biometrics: Theory, Applications, and Systems*, pages 1–6. IEEE, 2007.

[26] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980.

[27] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 249–256, 2010.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning. 6.2.2.3 Softmax Units for Multinoulli Output Distributions.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[30] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision.* Cambridge University Press, 2003.

[31] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction.* Springer, 2 edition, 2009.

[32] Munawar Hayat, Salman H. Khan, Naoufel Werghi, and Roland Goecke. Joint registration and representation learning for unconstrained face identification. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1551–1560, 2017.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[35] Lingxiao He, Haiqing Li, Qi Zhang, Zhenan Sun, and Intelligence Technology. Dynamic feature learning for partial face recognition. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[36] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[37] David H. Hubel and Torsten N. Wiesel. Receptive fields of single neurons in the cat's striate cortex. *Journal of Physiology*, 148:574–591, 1959.

[38] David H. Hubel and Torsten N. Wiesel. Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex. *Journal of Physiology*, 160:106–154, 1962.

[39] Patrik Huber, Guosheng Hu, Rafael Tena, Pouria Mortazavian, and Willem P Koppen. A multiresolution 3d morphable face model and fitting framework. *Proceedings of the 11th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, 2015.

[40] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. volume 37 of *Proceedings of Machine Learning Research*, pages 448–456, Lille, France, 07–09 Jul 2015. PMLR.

[41] Y Kaya and K Kobayashi. A basic study on human face recognition. In *Frontiers of Pattern Recognition*, pages 265–289. Elsevier, 1972.

[42] Ira Kemelmacher-Shlizerman, Steven M Seitz, Daniel Miller, and Evan Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4873–4882, 2016.

[43] Davis E King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 10(Jul):1755–1758, 2009.

[44] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 2014. cite arxiv:1412.6980 Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[45] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural*

Information Processing Systems 25, pages 1097–1105. Curran Associates, Inc., 2012.

[46] Amit Kumar and Rama Chellappa. Disentangling 3D pose in a dendritic CNN for unconstrained 2D face alignment. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[47] Ludmila I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms.* Wiley-Interscience, New York, NY, USA, 2004.

[48] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 239–248. IEEE, 2016.

[49] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[50] Yann LeCun et al. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[51] Jose Lezama, Qiang Qiu, and Guillermo Sapiro. Not afraid of the dark: NIR-VIS face recognition via cross-spectral hallucination and low-rank embedding. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[52] Beiwen Li, Yajun Wang, Junfei Dai, William Lohry, and Song Zhang. Some recent advances on superfast 3d shape measurement with digital binary defocusing techniques. *Optics and Lasers in Engineering*, 54:236–246, 2014.

[53] Haoliang Li, Peisong He, Shiqi Wang, Anderson Rocha, Xinghao Jiang, and Alex C Kot. Learning generalized deep feature representation for face anti-spoofing. *IEEE Transactions on Information Forensics and Security*, 13(10):2639–2652, 2018.

[54] Jiangwei Li, Yunhong Wang, Tieniu Tan, and Anil K Jain. Live face detection based on the analysis of fourier spectra. In *Biometric Technology for Human Identification*, volume 5404, pages 296–304. International Society for Optics and Photonics, 2004.

[55] Lei Li, Xiaoyi Feng, Xiaoyue Jiang, Zhaoqiang Xia, and Abdenour Hadid. Face anti-spoofing via deep local binary patterns. *International Conference in Image Processing, IEEE.*, 2017.

[56] Fayao Liu, Chunhua Shen, and Guosheng Lin. Deep convolutional neural fields for depth estimation from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5162–5170, 2015.

[57] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. SphereFace: deep hypersphere embedding for face recognition. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 212–220, 2017.

[58] Yaojie Liu, Amin Jourabloo, and Xiaoming Liu. Learning deep models for face anti-spoofing: Binary or auxiliary supervision. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 389–398, 2018.

[59] Yu Liu, Fangyin Wei, Jing Shao, Lu Sheng, Junjie Yan, Xiaogang Wang, and Sensetime Group Limited. Exploring disentangled feature representation beyond face identification. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2080–2089, 2018.

[60] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[61] Mortimer Marks. System and devices for time delay 3d, September 29 1992. US Patent 5,151,821.

[62] Brianna Maze, Jocelyn Adams, James Duncan, Nathan Kalka, Tim Miller, Charles Otto, Anil Jain, W. Niggel, Janet Anderson, Jordan Cheney, and Patrick Grother. Iarpa janus benchmark - c: Face dataset and protocol. In *2018 International Conference on Biometrics (ICB)*, pages 158–165, 02 2018.

[63] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[64] Tom Mitchell. *Machine Learning.* McGraw-Hill Education, 1997.

[65] Kevin Patrick Murphy. *Machine Learning: A Probabilistic Perspective.* MIT Press, 2012.

[66] Aaron Nech and Ira Kemelmacher-Shlizerman. Level playing field for million scale face recognition. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7044–7053, 2017.

[67] Hong-Wei Ng and Stefan Winkler. A data-driven approach to cleaning large face datasets. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 343–347. IEEE, 2014.

[68] International Conference on Computational Photography 2020. Session 3.a iccp 2020, 2020.

[69] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *BMVC*, 2015.

[70] Stefano Pini, Filippo Grazioli, Guido Borghi, Roberto Vezzani, Reggio Emilia, and Rita Cucchiara. Learning to generate facial depth maps. *arXiv preprint arXiv:1805.11927*, 2018.

[71] Emmanuel Prados and Olivier Faugeras. Shape from shading. In *Handbook of Mathematical Models in Computer Vision*, pages 375–388. Springer, 2006.

[72] Mark D Pritt and Dennis C Ghiglia. *Two-Dimensional Phase Unwrapping: Theory, Algorithms, and Software.* Wiley, 1998.

[73] Dikpal Reddy, Amit Agrawal, and Rama Chellappa. Enforcing integrability by error correction using l1-minimization. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2350–2357. IEEE, 2009.

[74] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[75] Guy Rosman and Daniela Rus. Information-driven adaptive structured-light scanners. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[76] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):824–840, 2009.

[77] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

[78] Laurent Schwartz. *Theorie des Distributions.* Hermann, 1957.

[79] Amnon Shashua. On photometric issues in 3d visual recognition from a single 2d image. *International Journal of Computer Vision*, 21(1-2):99–122, 1997.

[80] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[81] Lawrence Sirovich and M Kirby. Low-dimensional procedure for the characterization of human faces. *Journal of the Optical Society of America. A, Optics and Image Science*, 4:519–24, 04 1987.

[82] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[83] Yi Sun, Ding Liang, Xiaogang Wang, and Xiaoou Tang. Deepid3: Face recognition with very deep neural networks. *CoRR*, abs/1502.00873, 2015.

[84] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. *CoRR*, abs/1406.4773, 2014.

[85] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation from predicting 10,000 classes. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1891–1898, 2014.

[86] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deeply learned face representations are sparse, selective, and robust. *CoRR*, abs/1412.1265, 2014.

[87] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. *CoRR*, abs/1602.07261, 2016.

[88] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[89] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

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

[91] Mitsuo Takeda and Kazuhiro Mutoh. Fourier transform profilometry for the automatic measurement of 3-d object shapes. *Applied Optics*, 22(24):3977–3982, 1983.

[92] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *CoRR*, abs/1905.11946, 2019.

[93] Luan Tran, Xi Yin, and Xiaoming Liu. Disentangled representation learning GAN for pose-invariant face recognition. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[94] Jack Tumblin, Amit Agrawal, and Ramesh Raskar. Why i want a gradient camera. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, pages 103–110. IEEE, 2005.

[95] Stanford University. Cs231n: Convolutional neural networks for visual recognition. https://cs231n.github.io/convolutional-networks/, 2020. Accessed: 2020-10-18.

[96] Paul Viola and Michael Jones. Robust real-time object detection. In *International Journal of Computer Vision*, 2001.

[97] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. CosFace: Large margin cosine loss for deep face recognition. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5265–5274, 2018.

[98] Lior Wolf, Tal Hassner, and Itay Maoz. Face recognition in unconstrained videos with matched background similarity. In *CVPR 2011*, pages 529–534, 2011.

[99] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[100] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[101] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Faceness-net: Face detection through deep facial part responses. *CoRR*, abs/1701.08393, 2017.

[102] Chun-Hsiao Yeh and Herng-Hua Chang. Face liveness detection with feature discrimination between sharpness and blurriness. In *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*, pages 398–401. IEEE, 2017.

[103] Xin Yu and Fatih Porikli. Hallucinating very low-resolution unaligned and noisy face images by transformative discriminative autoencoders. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[104] Stefanos Zafeiriou, Gary A Atkinson, Mark F Hansen, William A P Smith, Vasileios Argyriou, Maria Petrou, Melvyn L Smith, and Lyndon N Smith. Face recognition and verification using photometric stereo: The photoface database and a comprehensive evaluation. *IEEE Transactions on Information Forensics and Security*, 8(1):121–135, 2013.

[105] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

[106] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multi-task cascaded convolutional networks. *CoRR*, abs/1604.02878, 2016.

[107] Song Zhang. Recent progresses on real-time 3d shape measurement using digital fringe projection techniques. *Optics and Lasers in Engineering*, 48(2):149–158, 2010.

[108] Song Zhang. *Handbook of 3D Machine Vision: Optical Metrology and Imaging*. CRC press, 2013.

[109] Song Zhang and Shing-Tung Yau. High-resolution, real-time 3d absolute coordinate measurement based on a phase-shifting method. *Optics Express*, 14(7):2644–2649, 2006.

[110] Xu Zhang, Xiyuan Hu, Chen Chen, and Silong Peng. Face spoofing detection based on 3D lighting environment analysis of image pair. *IEEE International Conference on Pattern Recognition (ICPR)*, 2016.

[111] Zhiwei Zhang, Junjie Yan, Sifei Liu, Zhen Lei, Dong Yi, and Stan Z Li. A face antispoofing database with diverse attacks. In *International Conference on Biometrics (ICB)*, pages 26–31. IEEE, 2012.

[112] Jian Zhao, Yu Cheng, Yan Xu, Lin Xiong, Jianshu Li, Fang Zhao, Karlekar Jayashree, Sugiri Pranata, Shengmei Shen, Junliang Xing, Shuicheng Yan, and Jiashi Feng. Towards pose invariant face recognition in the wild. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[113] Wenyi Zhao, Rama Chellappa, P Jonathon Phillips, and Azriel Rosenfeld. Face recognition: A literature survey. *ACM Computing Surveys (CSUR)*, 35(4):399–458, 2003.

[114] Yutong Zheng, Dipan K Pal, and Marios Savvides. Ring loss: Convex feature normalization for face recognition. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5089–5097, 2018.

[115] Shaohua Kevin Zhou, Gaurav Aggarwal, Rama Chellappa, and David W Jacobs. Appearance characterization of linear lambertian objects, generalized photometric stereo, and illumination-invariant face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(2):230–245, 2007.

[116] Xuan Zou, Josef Kittler, Kieron Messer, and United Kingdom. Face recognition using active near-IR illumination. *British Machine Vision Conference (BMVC)*, 2005.

[117] Syed Zulqarnain Gilani and Ajmal Mian. Learning from millions of 3D scans for large-scale 3D face recognition. pages 1895–1905, 2018.

List of tables

Esta página ha sido intencionalmente dejada en blanco.

List of figures

Esta es la última página.
Compilado el domingo 1 agosto, 2021.
`http://iie.fing.edu.uy/`