



# UNIVERSIDAD DE GRANADA

Facultad de Ciencias

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y  
MATEMÁTICAS

TRABAJO DE FIN DE GRADO

## Localización de landmarks cefalométricos por medio de técnicas de few-shot learning y análisis de redes convolucionales

Presentado por:  
Alejandro Borrego Megías

Tutores:  
Pablo Mesejo Santiago  
*DECSAI*

Javier Merí de la Maza  
*Dpto Análisis Matemático*

Mentor:  
Guillermo Gómez Trenado  
*DECSAI*

Curso académico 2022-2023



# Localización de landmarks cefalométricos por medio de técnicas de few-shot learning y análisis de redes convolucionales

Alejandro Borrego Megías

Alejandro Borrego Megías *Localización de landmarks cefalométricos por medio de técnicas de few-shot learning y análisis de redes convolucionales.*  
Trabajo de fin de Grado. Curso académico 2022-2023.

**Responsables de tutorización**

Pablo Mesejo Santiago  
*DECSAI*

Doble Grado en Ingeniería Informática y Matemáticas

**Responsable de mentorización**

Javier Merí de la Maza  
*Dpto Análisis Matemático*  
Guillermo Gómez Trenado  
*DECSAI*

Facultad de Ciencias  
Universidad de Granada

**DECLARACIÓN DE ORIGINALIDAD**

D./Dña. Alejandro Borrego Megías

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2022-2023, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 6 de noviembre de 2022

Fdo: Alejandro Borrego Megías



## **Agradecimientos**

Gracias a mi familia y a mi pareja por ser un apoyo fundamental y permitir las condiciones idóneas en casa para la realización de este trabajo. Gracias a Javier, Pablo y Guillermo por darme la oportunidad de trabajar con ellos y por haberme ayudado durante todo el proceso.



# Índice general

Agradecimientos	VII
Abstract	XIII
Resumen	XV
<b>I. Análisis de Redes Convolucionales</b>	<b>1</b>
1. Introducción	3
2. Modelización Matemática de una Red Neuronal Convolutacional	7
2.1. De Fourier a las ondeletas de Littlewood-Paley . . . . .	7
2.1.1. El módulo de la Transformada de Fourier . . . . .	7
2.1.2. Alternativa: Las ondeletas . . . . .	12
2.1.3. La Transformada de Littlewood-Paley . . . . .	16
2.1.4. Convenios para futuras secciones . . . . .	19
2.2. El operador de dispersión sobre un camino ordenado . . . . .	20
2.2.1. Ejemplo para obtener coeficientes invariantes por traslaciones . . . . .	21
2.2.2. El operador módulo . . . . .	22
2.2.3. Definición de camino de frecuencias y propiedades. . . . .	23
2.2.4. Construcción del operador de dispersión. . . . .	24
2.3. Propagador de dispersión y conservación de la Norma . . . . .	26
2.3.1. Proceso de dispersión del propagador. . . . .	26
2.3.2. Diferencias y similitudes con una CNN . . . . .	27
2.3.3. Relación con herramientas clásicas de visión por computador . . . . .	28
2.3.4. Operador no expansivo. . . . .	28
2.3.5. Conservación de la norma. . . . .	29
2.3.6. Conclusiones extraídas del teorema . . . . .	33
3. Invarianza por Traslaciones	35
3.1. No expansividad del operador de ventana en conjuntos de caminos . . . . .	35
3.2. Invarianza por traslaciones . . . . .	38
4. Conclusiones y trabajos futuros	43
<b>II. Localización de landmarks cefalométricos por medio de técnicas de few-shot learning</b>	<b>45</b>
5. Introducción	47
5.1. Descripción del problema y Motivación . . . . .	47
5.1.1. Base de datos proporcionada . . . . .	49

## *Índice general*

5.2.	Requisitos mínimos del algoritmo . . . . .	52
5.3.	Objetivos . . . . .	53
5.4.	Planificación . . . . .	53
<b>6.</b>	<b>Fundamentos Teóricos y Métodos</b>	<b>57</b>
6.1.	Aprendizaje Automático . . . . .	57
6.1.1.	Aprendizaje Supervisado . . . . .	58
6.1.2.	Aprendizaje no Supervisado . . . . .	60
6.1.3.	Aprendizaje Automático en este Trabajo . . . . .	60
6.1.4.	Visión por Computador . . . . .	61
6.1.5.	Deep Learning . . . . .	62
6.2.	Redes Neuronales Convolucionales . . . . .	65
6.2.1.	Capa Convolucional . . . . .	67
6.2.2.	Capa de Pooling . . . . .	68
6.2.3.	Capa Totalmente Conectada (Fully Connected) . . . . .	69
6.2.4.	Batch Normalization . . . . .	69
6.2.5.	Optimizador Adam . . . . .	70
6.2.6.	Proceso de entrenamiento de una CNN . . . . .	70
6.2.7.	Evolución de las CNN . . . . .	70
6.3.	Autoencoders . . . . .	74
6.3.1.	Introducción . . . . .	74
6.3.2.	Evolución de los Autoencoders . . . . .	75
6.4.	Técnicas empleadas . . . . .	79
6.4.1.	Few-shot Learning y Data Augmentation . . . . .	79
<b>7.</b>	<b>Estado del Arte</b>	<b>81</b>
7.1.	Localización de landmarks cefalométricos en imágenes . . . . .	81
7.1.1.	Evolución en la identificación forense de landmarks cefalométricos . . . . .	81
<b>8.</b>	<b>Implementación</b>	<b>87</b>
8.1.	Diseño del Software . . . . .	87
8.2.	Entorno de ejecución . . . . .	89
<b>9.</b>	<b>Solución propuesta y experimentos realizados</b>	<b>93</b>
9.1.	Framework empleado: 3FabRec . . . . .	94
9.1.1.	Arquitectura Adversarial Autoencoder . . . . .	95
9.1.2.	Función de pérdida . . . . .	99
9.1.3.	Proceso de entrenamiento de la red . . . . .	101
9.1.4.	Bases de datos usadas por el framework . . . . .	102
9.2.	Métricas . . . . .	102
9.2.1.	Métricas usadas en el entrenamiento . . . . .	102
9.2.2.	Métricas empleadas en validación y testing . . . . .	103
9.3.	Preprocesamiento de los datos . . . . .	104
9.3.1.	Identificación de caras en las imágenes . . . . .	104
9.3.2.	Creación del fichero annotations en el dataset . . . . .	106
9.3.3.	Reajuste de los bounding boxes . . . . .	107
9.3.4.	Separación en conjuntos de entrenamiento y validación . . . . .	110

9.4.	Experimentación . . . . .	111
9.4.1.	Hipótesis iniciales . . . . .	111
9.4.2.	Modelo Base . . . . .	111
9.4.3.	Modelo con reentrenamiento del encoder . . . . .	114
9.4.4.	Modelo con reentrenamiento del decoder . . . . .	117
9.4.5.	Modelo con Data Augmentation . . . . .	119
9.4.6.	Elección de modelo y obtención de resultados . . . . .	121
9.5.	Comparativa entre 3FabRec y HyperFace-ResNet101 . . . . .	124
9.5.1.	Preprocesamiento y base de datos empleada . . . . .	124
9.5.2.	Métricas empleadas . . . . .	125
9.5.3.	Comparación de resultados . . . . .	125
<b>10.</b>	<b>Conclusiones y Trabajos Futuros</b>	<b>127</b>
10.1.	Objetivos Satisfechos . . . . .	127
10.2.	Trabajos Futuros y comentarios . . . . .	128
<b>A.</b>	<b>Apéndice A</b>	<b>131</b>
A.1.	Resultados por imagen durante el entrenamiento del modelo base . . . . .	131
<b>B.</b>	<b>Apéndice B</b>	<b>137</b>
B.1.	Resultados por imagen durante el entrenamiento del modelo de entrenamiento del encoder . . . . .	137
<b>C.</b>	<b>Apéndice C</b>	<b>143</b>
C.1.	Resultados por imagen durante el entrenamiento del modelo de entrenamiento del decoder . . . . .	143
<b>D.</b>	<b>Apéndice D</b>	<b>149</b>
D.1.	Resultados por imagen durante el entrenamiento del modelo con data augmentation . . . . .	149
	<b>Bibliografía</b>	<b>155</b>



## Abstract

The main purpose of the work detailed below is to show a possible mathematic modelization of a Convolutional Neural Network and demonstrate one of the main properties of this kind of network, Translation Invariance. On the other side, we will adapt the architecture of an existing network in order to solve a real problem about cephalometric landmark detection from a Machine Learning perspective using few-shot learning.

Convolutional Neural Networks are recent tools which have demonstrated a great capacity for image-processing tasks. Their good performance on this kind of tasks can be checked empirically. However, their mathematical modelization and the theoretical justification of why they are good for this kind of tasks still is an open case study. For this reason, the main purpose of this part of the work is to present a possible modelization for CNN based on the theoretical approximation presented by Mallat in his work **Group Invariant Scattering**. Once the modelization is shown, we try to prove one of the most important properties of this network, Translation Invariance.

To do so, we present an operator called scattering propagator. Using this operator in a cascade of convolutions we reach the windowed scattering transform, presented as the mathematical modelization of CNN. We present a set of properties that this operator must satisfy, like Lipschitz-continuity, non-expansivity and translation invariant coefficients. The operator that satisfies all these properties is the Littlewood-Paley wavelet transform. Once we have the operator we define the modelization of the windowed scattering and we study the similarities with the Convolutional Neural Networks. To prove the translation invariance we use that the windowed operator is non-expansive.

In the second part of this work, we adapt an existing CNN specialized in facial landmarks detection called 3FabRec (which is an Adversarial Autoencoder with interleaved layers that predict landmarks). The objective is to predict cephalometric facial landmarks using this framework. Cephalometric landmarks have biological inspiration. The main problem is the small dataset provided with only a few images in-the-wild to train the model.

To do so, the dataset provided had 167 images of people in the wild. We wanted to train the framework to predict a maximum of 30 landmarks in each image. After a previous analysis of the dataset, we discovered that not all the landmarks are marked in all images, and we needed to use an auxiliary network called Facenet to identify bounding boxes on the images and be able to train the model. Due to the few amount of data we have, this is contained in the field of few-shot learning.

We studied the state-of-art discovering that only a few articles in the search we did were related to this specific problem. In these articles, a set of images in the same position and illumination conditions were used, instead of the in-the-wild dataset we use. Our model solves the problem using a Network pretrained on the AFLW dataset and making fine-tuning with the forensic dataset. We apply data augmentation techniques to do so and we obtain competitive results like an average NME of 2.65.

*Abstract*

Finally, we compare our results with the model proposed by Guillermo Gómez in his project on 2019. The median RMSE of our model is 5.3862 and in the model presented in 2019 is 3.4106. Despite the lower average RMSE error, our model improves the performance in some landmarks using a smaller dataset.

## Resumen

El trabajo fin de grado (TFG) que a continuación se detalla tiene como objetivo presentar una posible modelización, desde el punto de vista matemático, de lo que es una **Red Neuronal Convolucional** (CNN), junto con la demostración de una de sus principales propiedades: la invarianza frente a traslaciones. Por otro lado, se pretende adaptar una arquitectura de red neuronal convolucional ya existente a un problema real que resulte apropiado con *few-shot* learning, que se abordará desde el punto de vista informático.

Las CNN son herramientas, que pese a haber surgido recientemente, han demostrado tener un gran potencial para el procesamiento de imágenes, convirtiéndose rápidamente en una de las principales herramientas para estos fines. Su buen rendimiento en este tipo de tareas es comprobable empíricamente, sin embargo siguen siendo una vía de estudio abierta en lo que se refiere la modelización matemática y la justificación teórica de estos resultados. Así, en la primera parte del TFG se pretende desarrollar la **modelización matemática** propuesta por Mallat, en su trabajo **Group Invariant Scattering**, junto con la demostración de una de sus principales propiedades: **la invarianza por traslaciones**.

En lo que respecta a la modelización, comenzamos con la búsqueda de un operador denominado *propagador de dispersión*, con el cual, aplicando la operación de convolución de forma recursiva, se construye el denominado *operador de ventana*. Éste, será la modelización propuesta y actuará sobre caminos de frecuencias finitos (pues, como demostraríamos, se puede conseguir retener tanta información como se quiera). Algunas propiedades importantes que debe verificar nuestro operador son la *Lipschitz-continuidad*, que sea no expansivo o que produzca coeficientes invariantes a traslaciones. Esto, nos impedirá usar herramientas clásicas como la transformada de Fourier, y tendremos que recurrir a la transformada de ondeletas de *Littlewood-Paley*. Al llegar a la posible modelización se realiza un estudio de las diferencias y similitudes que guarda con las CNN tal y como las conocemos.

En lo que respecta a la invarianza por traslaciones, haciendo uso de las propiedades del *operador de ventana*, se consigue demostrar, en primer lugar, que dicho operador es no expansivo al aplicarse en conjuntos de caminos de frecuencias, y con dicha propiedad se demuestra la invarianza por traslaciones del mismo.

En la segunda parte del trabajo, se pretende adaptar un framework existente especializado en el reconocimiento de landmarks faciales, denominado **3FabRec** (un *Adversarial Autoencoder* con capas auxiliares encargadas del reconocimiento de landmarks en imágenes) para que sea capaz de marcar landmarks cefalométricos, puntos antropométricos situados en la cabeza, empleando un conjunto de datos reducido y con imágenes, en diversas posiciones y condiciones de calidad e iluminación.

Para llevar a cabo esta tarea se parte de un conjunto de 167 imágenes de distintos sujetos en una gran variedad de posturas, condiciones de iluminación y de calidad de imagen con los landmarks etiquetados por un experto. Este conjunto de datos se dividirá en entrenamiento y test. Entrenaremos la red neuronal mencionada anteriormente para lograr predecir hasta un máximo de **30 landmarks** distribuidos por toda la cara. Tras un análisis exhaustivo, vemos

## *Resumen*

como no todos los landmarks se encuentran marcados en todas las imágenes del dataset o la necesidad de emplear un identificador de caras para marcar *bounding boxes* en las imágenes y poder recortarlas en un paso previo al entrenamiento de la red. Debido a los pocos ejemplos de entrenamiento que poseemos, consideramos el problema dentro del ámbito del *few-shot learning*.

El problema en cuestión que se trata ha sido tratado en anteriores TFGs, como por ejemplo el de Guillermo Gómez de 2019 que usaremos para comparar los resultados obtenidos. No obstante, se ha revisado con detenimiento la literatura existente y el estado del arte en este campo, descubriendo que se trata de una aproximación novedosa. Los trabajos encontrados que guardan relación sobre el tema son muy escasos, y los que aplican técnicas de deep learning, no emplean conjuntos de datos *in-the-wild*. Además, nuestra propuesta plantea aprovechar el conocimiento adquirido por una red en un amplio dataset, como es AFLW, en el reconocimiento de landmarks no antropométricos y ajustarla mediante *fine-tuning* y técnicas *data augmentation* para el reconocimiento de landmarks cefalométricos. Vamos a resolver, por tanto, un problema de **regresión** y como veremos, los resultados obtenidos son competitivos con el estado del arte, obteniéndose una media de error NME de 2.65.

Por otro lado, se realiza una comparación a nivel cuantitativo entre la solución propuesta y el presentado en el TFG de 2019. La mediana del RMSE cometido por nuestro modelo es de 5.3862 píxeles de error, frente a los 3.4106 del otro modelo. No obstante veremos como a nivel de landmark, se mejora el rendimiento en el marcado de ciertos puntos y se ha obtenido con un conjunto menor de datos etiquetados.

# **Parte I.**

## **Análisis de Redes Convolucionales**



# 1. Introducción

Actualmente, las **Redes Neuronales Convolucionales** (CNN<sup>1</sup>) son una de las herramientas más usadas de la Inteligencia Artificial. De hecho, son el principal objeto de estudio del **Deep Learning**, una rama del *Aprendizaje Automático* (AA) en la que hoy en día se está invirtiendo mucho esfuerzo en investigar y de la que, anualmente, se publican muchos artículos. Destaca, especialmente, el excelente desempeño que tienen en el procesamiento de imágenes para tareas de clasificación, segmentación o incluso generación de nuevas imágenes. Es por ello que, en el presente trabajo, nos proponemos intentar realizar una **modelización matemática** de este tipo de redes, para conocerlas mejor desde un punto de vista más teórico y poder demostrar una de sus principales propiedades: **la invarianza por traslaciones**.

Uno de los principales problemas en el ámbito del AA y la visión por computador es el de clasificación de imágenes y detección de objetos en las mismas. En este contexto, definimos el concepto de **invarianza** como la capacidad de reconocer un objeto en una imagen incluso si su apariencia ha variado en algún sentido (mediante una rotación, una ligera deformación o una traslación). Esto es algo muy importante, pues nos indica que se preserva la identidad del objeto incluso a pesar de haberse sometido a ciertos cambios.

De esta forma definimos la **invarianza por traslaciones** como la capacidad de reconocer la identidad de un objeto en una imagen incluso si este se ha desplazado (véase la [Figura 1.1](#)). Esta propiedad es fundamental y sabemos que las CNN la verifican.

## Invarianza por traslaciones

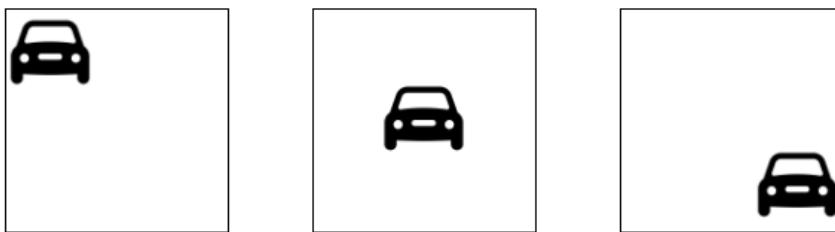


Figura 1.1.: Los tres coches deben identificarse como iguales, aunque se encuentren desplazados.

Otra propiedad importante es la **invarianza frente a pequeñas deformaciones** (difeomorfismos, véase la [Figura 1.2](#)), que es la capacidad de reconocer la identidad de un objeto en una imagen a pesar de que este pueda haber sido alterado con pequeñas deformaciones (véase la [Figura 1.3](#) y [Figura 1.4](#)).

Realizaremos nuestro estudio sobre las funciones  $L^2(\mathbb{R}^d)$ . En este espacio buscaremos un

---

<sup>1</sup>Convolutional Neural Network

## 1. Introducción



Figura 1.2.: Acción de un difeomorfismo en una rejilla.



Figura 1.3.: Todas las imágenes deberían clasificarse como 5, pese a las deformaciones.



Figura 1.4.: Deformación excesiva que permite confundir el 1 con el 2 cuando se le aplica el difeomorfismo. Por eso nos centramos en “pequeñas” deformaciones, para no alterar la identidad del objeto en la imagen.

operador que sea Lipschitz-continuo por la acción de difeomorfismos y que mantenga información de alta frecuencia para diferenciar entre distintos tipos de señales.

La invarianza por translaciones, entendida en el contexto de las imágenes puede verse como trasladar cada pixel de la imagen en una misma dirección la misma distancia. En este sentido

definimos las traslaciones de funciones de cuadrado integrable:

**Definición 1.0.1.**  $L_c f(x) = f(x - c)$  es la traslación de  $f \in L^2(\mathbb{R}^d)$  por  $c \in \mathbb{R}^d$ .

Así, decimos que un operador  $\Phi$  sobre  $L^2(\mathbb{R}^d)$ , es invariante por traslaciones si  $\Phi(L_c f(x)) = \Phi(f)$  para todo  $f \in L^2(\mathbb{R}^d)$  y para todo  $c \in \mathbb{R}^d$ . En el siguiente apartado trataremos el caso del módulo de la transformada de Fourier de  $f$  como un ejemplo de un operador invariante por traslaciones, aunque la aparición de inestabilidades frente a deformaciones en las altas frecuencias nos obligará a descartarlo como opción para la modelización de CNN, pues no preserva la Lipschitz-continuidad frente a difeomorfismos.

Para preservar la estabilidad en  $L^2(\mathbb{R}^d)$  queremos que  $\Phi$  sea no-expansiva.

**Definición 1.0.2.** Decimos que  $\Phi$  es no-expansiva si:

$$\forall f, h \in L^2(\mathbb{R}^d) \quad \|\Phi(f) - \Phi(h)\| \leq \|f - h\|$$

donde  $\|f\|$  denota la norma de  $f$  en  $L^2(\mathbb{R}^d)$ .

**Definición 1.0.3.** Una función diferenciable  $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$ , es un *difeomorfismo* si  $f$  es una biyección y su inversa  $f^{-1} : \mathbb{R}^d \rightarrow X$  es también diferenciable.

En nuestro caso, vamos a encargarnos de verificar la Lipschitz-continuidad relativa a la acción de pequeños difeomorfismos cercanos a las traslaciones. Dichos difeomorfismos transforman  $x \in \mathbb{R}^d$  en  $x - \tau(x)$  donde  $\tau$  es el campo de desplazamiento.

**Definición 1.0.4.** Denotemos  $L_\tau f(x) = f(x - \tau(x))$  como la acción del difeomorfismo  $\mathbb{1} - \tau$  sobre  $f$ .

Recordemos que la condición de Lipschitz es la siguiente:

**Definición 1.0.5.** Sea  $f : M \rightarrow N$  una función entre dos espacios métricos  $M$  y  $N$  con sus respectivas distancias  $d_M$  y  $d_N$ . Se dice que  $f$  satisface la condición de Lipschitz si  $\exists C > 0$  tal que:

$$d_N(f(x), f(y)) \leq C d_M(x, y), \quad \forall x, y \in M$$

En nuestro caso, dado que el espacio de partida es  $L^2(\mathbb{R}^d)$  y los puntos que vamos a comparar son las funciones  $f$  y  $L_\tau f = f(x - \tau(x))$  sabemos que  $\|\Phi(f) - \Phi(L_\tau f)\|$  estará acotada por  $\|f\| d(\mathbb{1}, \mathbb{1} - \tau)$ , de manera que necesitamos definir una distancia entre el difeomorfismo  $\mathbb{1}$  y  $\mathbb{1} - \tau$ .

**Definición 1.0.6.** Se define una distancia entre  $\mathbb{1} - \tau$  y  $\mathbb{1}$  en cualquier subconjunto compacto  $\Omega$  de  $\mathbb{R}^d$  como

$$d_\Omega(\mathbb{1}, \mathbb{1} - \tau) = \sup_{x \in \Omega} |\tau(x)| + \sup_{x \in \Omega} |\nabla \tau(x)| + \sup_{x \in \Omega} |H\tau(x)|$$

Donde  $|\tau(x)|$  es la norma euclídea en  $\mathbb{R}^d$ ,  $|\nabla \tau(x)|$  es la norma del supremo de la matriz jacobiana  $\nabla \tau(x)$ , y  $|H\tau(x)|$  es la norma del supremo del Hessiano.

No obstante, debido a que trataremos con funciones que son invariantes por traslaciones, la condición de Lipschitz será independiente de la amplitud máxima de la traslación  $\sup_{x \in \mathbb{R}^d} |\tau(x)|$ , es por ello que obviaremos este término.

En lo que sigue vamos a denotar:

## 1. Introducción

- $\|\nabla \tau\|_\infty := \sup_{x \in \mathbb{R}^d} |\nabla \tau(x)|$
- $\|H\tau\|_\infty := \sup_{x \in \mathbb{R}^d} |H\tau(x)|$

Así, podemos finalmente expresar la condición de Lipschitz que un operador debería satisfacer en nuestro caso como:

**Definición 1.0.7.** Un operador invariante por traslaciones  $\Phi$  se dice “*Lipchitz-continuo*” por la acción de los difeomorfismos  $C^2$  si para cualquier compacto  $\Omega \subset \mathbb{R}^d$  existe una constante  $C$  tal que para todo  $f \in L^2(\mathbb{R}^d)$  con soporte en  $\Omega$  y para todo  $\tau \in C^2(\mathbb{R}^d)$  se cumple:

$$\|\Phi(f) - \Phi(L_\tau f)\|_{\mathcal{H}} \leq C\|f\|(\|\nabla \tau\|_\infty + \|H\tau\|_\infty). \quad (1.1)$$

La continuidad Lipschitz de (1.1) implica que  $\Phi$  es invariante por traslaciones globales, pero dicha condición es mucho más fuerte: (1.1) garantiza que  $\Phi$  se ve poco afectada por los términos de primer y segundo grado de difeomorfismos que son traslaciones locales.

Una vez presentadas las principales herramientas con las que trabajaremos, veremos en las futuras secciones cómo, para solucionar el problema, se optará por utilizar **transformadas de ondeletas**. Aunque esto abre nuevos frentes como el hecho de que estas **no son invariantes por traslaciones**. Para lograr la invarianza por traslaciones será necesario componer la transformada con un **operador no lineal** para obtener coeficientes invariantes. Este nuevo operador consistirá en una **cascada de convoluciones** de operadores no lineales y no commutativos de manera que cada uno de ellos calcula el módulo de la transformada de ondeletas, y será este nuevo operador el que podremos interpretar como la modelización matemática de una CNN.

## 2. Modelización Matemática de una Red Neuronal Convolucional

Nuestro primer objetivo será tratar de llegar a la modelización matemática de lo que es una CNN. Para ello, necesitamos definir un operador, que denominaremos **propagador de dispersión** (PD), que será el que aplicaremos de forma recursiva en una cascada de convoluciones. Explicaremos la problemática de elegir un operador *lipschitz-continuo* bajo la acción de difeomorfismos e *invariante por traslaciones* para evitar problemas como la inestabilidad en altas frecuencias que se producen en las señales bajo la acción de difeomorfismos.

En segundo lugar, veremos posibles alternativas para evitar que se produzcan estas inestabilidades mediante el uso de bases de la transformada de ondeletas de **Littlewood-Paley**. En concreto con esta alternativa obtendremos un operador **Lipschitz-continuo** bajo la acción de difeomorfismos.

Después, nuestra tarea será conseguir calcular coeficientes que sean invariantes por traslaciones, y para ello necesitaremos utilizar un operador no lineal como es el módulo.

Una vez tengamos un operador con todas las propiedades anteriores presentaremos el PD, y será la aplicación en cadena de este operador sobre un “*camino*” de frecuencias y rotaciones el que definirá la modelización matemática de una Red Neuronal Convolutinal. Todo este capítulo está basado en la investigación de Mallat y sigue como hilo conductor su publicación [Mal12] junto con otros autores que serán citados debidamente.

### 2.1. De Fourier a las ondeletas de Littlewood-Paley

#### 2.1.1. El módulo de la Transformada de Fourier

El análisis de Fourier, tradicionalmente, ha jugado un papel fundamental en el procesamiento de señales [Gon17], por lo que podría parecer un buen punto de partida para la construcción del *propagador de dispersión* emplear la **transformada de Fourier**, una de las herramientas matemáticas más potentes en este campo. La intuición detrás de su fórmula, es la de representar funciones no periódicas (pero que tienen área finita bajo su gráfica) como la integral de senos y cosenos multiplicados por una función que determina los pesos en cada instante. Formalmente, tiene la siguiente expresión:

$$\widehat{f}(\omega) := \int f(x)e^{-ix\omega} dx = \int f(x) [\cos x\omega - i \sin x\omega] dx.$$

Entre las propiedades más destacables de la transformada encontramos el hecho de que una función se puede recuperar sin pérdida de información a partir de su transformada de Fourier, lo cual nos permite poder trabajar en el “*Dominio de Fourier*”<sup>1</sup> ya que al calcular la integral, la función resultante sólo depende de  $\omega$  (la frecuencia), y posteriormente pasar de

<sup>1</sup>También llamado “*Dominio de Frecuencia*”

## 2. Modelización Matemática de una Red Neuronal Convolutacional

nuevo al dominio original de la función, aplicando la inversa de la transformada sin pérdida de información.

Esto, a priori, es algo atractivo, pues nos permitiría trabajar en un dominio más sencillo y extraer conclusiones que podemos traducir al dominio original de la señal sin pérdida de información. Además, en el estudio de señales se suele emplear el módulo de la transformada de Fourier para evitar fases complejas en el análisis. De esta forma, el operador que vamos a probar en primer lugar es:

**Definición 2.1.1.**  $\Phi(f) = |\hat{f}|$  módulo de la transformada de Fourier.

Vamos a comprobar si se trata de un operador válido para nuestro propósito. Para ello necesitamos en primer lugar que sea un operador **invariante por traslaciones**.

**Lema 2.1.2.** *El operador  $\Phi$  es invariante por traslaciones.*

*Demostración.* Consideramos, para cada  $c \in \mathbb{R}^d$ , la traslación  $L_c f(x) = f(x - c)$ . Se tiene que:

$$\widehat{L_c f}(w) = \int_{\mathbb{R}^d} L_c f(x) e^{-ixw} dx = \int_{\mathbb{R}^d} f(x - c) e^{-ixw} dx$$

y realizando el cambio de variable  $x - c = y$  se obtiene:

$$\begin{aligned} \int_{\mathbb{R}^d} f(x - c) e^{-ixw} dx &= \int_{\mathbb{R}^d} f(y) e^{-i(y+c)w} dy = \\ &= \int_{\mathbb{R}^d} f(y) e^{-iyw} e^{-icw} dy = \\ &= e^{-icw} \int_{\mathbb{R}^d} f(y) e^{-iyw} dy = e^{-icw} \widehat{f}(w) \end{aligned}$$

Por lo que se tiene que  $|\widehat{L_c f}(w)| = |e^{-icw}| |\widehat{f}(w)| = |\widehat{f}(w)|$  y entonces  $\Phi$  es invariante por traslaciones.  $\square$

Sin embargo, la invarianza por traslaciones no es suficiente. Necesitamos también que nuestro operador sea invariante frente a pequeñas deformaciones (difeomorfismos). De esta forma, diremos que un operador  $\Phi(f)$  es estable frente a deformaciones si verifica la Definición 1.0.7.

Consideramos la función  $\tau(x) := \epsilon x$  con  $0 < \epsilon \ll 1$ . De esta forma  $\|\nabla \tau(x)\|_\infty = \epsilon$  y  $\|H\tau(x)\|_\infty = 0$ . Con esto, la condición de Lipschitz para el módulo de la transformada de Fourier nos daría la existencia de una constante  $c > 0$  de modo que la desigualdad que se tendría que cumplir para cada  $f \in L^2(\mathbb{R}^d)$  y cada  $0 < \epsilon \ll 1$  sería:

$$\| |\hat{f}| - |\widehat{L_\tau f}| \| \leq c \|f\| (\|\nabla \tau\|_\infty + \|H\tau\|_\infty) = c \|f\| \epsilon. \quad (2.1)$$

Lo que implica encontrar una constante  $c \in \mathbb{R}$  que cumpla la desigualdad para cualquier valor de  $\epsilon$ . Vamos a ver un contraejemplo con una función de una dimensión por simplicidad.

Supongamos que tenemos  $f(x) = e^{i\zeta x} e^{-|x|}$  donde  $\zeta$  está por determinar. Calculamos ahora  $|\hat{f}|$  y  $|\widehat{L_\tau f}|$  teniendo en cuenta que :

$$\begin{aligned}
 |\widehat{f}(\omega)| &= \left| \int f(x) e^{-ix\omega} dx \right| \\
 &= \left| \int f(x) e^{-ix\omega} dx \right| \\
 &= \left| \int e^{i\xi x} e^{-|x|} e^{-ix\omega} dx \right| \\
 &= \left| \int e^{-ix(\xi-\omega)} dx \right| e^{-|x|} \\
 &= \left| \int e^{-|x|} [\cos x(\xi-\omega) - i \sin x(\xi-\omega)] dx \right|
 \end{aligned}$$

En el último paso podemos descomponer la integral en suma de dos, y para simplificar las operaciones llamamos  $\beta = (\xi - \omega)$ . Así, aplicando las siguientes fórmulas conocidas para el cálculo de integrales,

$$\int_{\mathbb{R}} \cos(\beta x) e^{-|x|} dx = \frac{2}{1 + \beta^2} \quad (2.2)$$

y

$$\int_{\mathbb{R}} \sin(\beta x) e^{-|x|} dx = 0 \quad (2.3)$$

a nuestro caso concreto, obtenemos que:

$$\begin{aligned}
 |\widehat{f}(\omega)| &= \left| \int \cos(x\beta) e^{-|x|} dx - i \int \sin(x\beta) e^{-|x|} dx \right| \\
 &= \frac{2}{1 + \beta^2} \\
 &= \frac{2}{1 + (\xi - \omega)^2}.
 \end{aligned}$$

Ahora pasamos a calcular  $|\widehat{L_\tau f}|$ :

$$\begin{aligned}
 |\widehat{L_\tau f}(\omega)| &= |\widehat{f}((1-\epsilon)\omega)| \\
 &= \left| \int f((1-\epsilon)x) e^{-ix\omega} dx \right| \\
 &= \left| \int e^{i\xi(1-\epsilon)x} e^{-|(1-\epsilon)x|} e^{-ix\omega} dx \right|.
 \end{aligned}$$

Ahora realizamos el siguiente cambio de variable

$$\tilde{x} = (1-\epsilon)x \implies x = \frac{\tilde{x}}{1-\epsilon}$$

2. Modelización Matemática de una Red Neuronal Convolucional

$$d\tilde{x} = (1 - \epsilon)dx \implies dx = \frac{1}{(1 - \epsilon)}d\tilde{x}$$

y aplicando los cambios a lo que teníamos nos queda

$$\begin{aligned} |\widehat{L_\tau f}(\omega)| &= \frac{1}{(1 - \epsilon)} \left| \int f((1 - \epsilon)x)e^{-ix\omega} dx \right| \\ &= \frac{1}{(1 - \epsilon)} \left| \int e^{i\xi\tilde{x}} e^{-|\tilde{x}|} e^{-i\frac{\tilde{x}}{(1-\epsilon)}\omega} d\tilde{x} \right| \\ &= \frac{1}{(1 - \epsilon)} \left| \int e^{i\left[\frac{(1-\epsilon)\xi-\omega}{(1-\epsilon)}\right]\tilde{x}} e^{-|\tilde{x}|} d\tilde{x} \right| \\ &= \frac{1}{(1 - \epsilon)} \left| \int e^{i\tilde{\beta}\tilde{x}} e^{-|\tilde{x}|} d\tilde{x} \right|. \end{aligned}$$

Como podemos ver, llegamos a una integral que se resuelve de la misma manera que en el caso anterior haciendo uso de (2.2) y (2.3):

$$\begin{aligned} |\widehat{L_\tau f}(\omega)| &= \frac{1}{(1 - \epsilon)} \frac{2}{1 + \tilde{\beta}^2} \\ &= \frac{1}{(1 - \epsilon)} \frac{2}{1 + \left[ \frac{(1-\epsilon)\xi-\omega}{(1-\epsilon)} \right]^2}. \end{aligned}$$

De esta forma hemos obtenido que para nuestro caso concreto de  $f(x) = e^{i\xi x} e^{-|x|}$ ,

$$\begin{aligned} \left\| |\widehat{L_\tau f}| - |\widehat{f}| \right\| &= \left\| \frac{1}{(1 - \epsilon)} \frac{2}{1 + \left[ \frac{(1-\epsilon)\xi-\omega}{(1-\epsilon)} \right]^2} - \frac{2}{1 + (\xi - \omega)^2} \right\| \\ &= 2 \left( \int_{\mathbb{R}} \left| \frac{\frac{1}{(1-\epsilon)}}{1 + \left[ \frac{(1-\epsilon)\xi-\omega}{(1-\epsilon)} \right]^2} - \frac{1}{1 + (\xi - \omega)^2} \right|^2 d\omega \right)^{1/2}. \end{aligned}$$

A continuación vamos a intentar aproximar el valor del módulo de la integral, para ello en primer lugar vamos a realizar el siguiente cambio de variable

$$\begin{aligned} t &= \omega - \xi \implies \omega - (1 - \epsilon)\xi = \omega - \xi + \epsilon\xi = t + \epsilon\xi \\ dt &= d\omega. \end{aligned}$$

Así, obtenemos que:

$$\begin{aligned} \int_{\mathbb{R}} \left| \frac{1}{1 + (\xi - \omega)^2} - \frac{\frac{1}{(1-\epsilon)}}{1 + \left[ \frac{(1-\epsilon)\xi - \omega}{(1-\epsilon)} \right]^2} \right|^2 d\omega &= \int_{\mathbb{R}} \left( \frac{1}{1 + (\xi - \omega)^2} - \frac{\frac{1}{(1-\epsilon)}}{1 + \left[ \frac{(1-\epsilon)\xi - \omega}{(1-\epsilon)} \right]^2} \right)^2 d\omega \\ &= \int_{\mathbb{R}} \left( \frac{1}{1 + t^2} - \frac{\frac{1}{(1-\epsilon)}}{1 + \left[ \frac{t + \epsilon\xi}{(1-\epsilon)} \right]^2} \right)^2 dt \end{aligned}$$

Representando la gráfica de  $g_1(t) = \frac{1}{1+t^2}$ , podemos ver cómo el valor de su integral se acumula en torno al origen de coordenadas, y en cambio  $g_2(t) = \frac{\frac{1}{(1-\epsilon)}}{1 + \left[ \frac{t + \epsilon\xi}{(1-\epsilon)} \right]^2}$  es una traslación y escalado de la función anterior. De esta forma, si  $\epsilon\xi$  es muy grande, el área encerrada por la función  $g_2(t)$  será prácticamente cero en la región del espacio donde  $g_1(t)$  concentra su integral. Dicho de otra forma, las dos funciones tendrían soporte “*casi disjunto*”. Véase la [Figura 2.1](#).

De esta forma, podemos tomar una constante  $M > 0$  tal que para un valor de  $\xi$  elevado se cumpla que:

$$\begin{aligned} \left| \int_{\mathbb{R}} \frac{1}{1 + (\xi - \omega)^2} - \frac{1}{1 + \left[ \frac{(1-\epsilon)\xi - \omega}{(1-\epsilon)} \right]^2} \right|^2 d\omega &\geq \int_{-M}^M (g_1(t) - g_2(t))^2 dt \\ &\approx \int_{-M}^M g_1(t)^2 dt. \end{aligned}$$

Y como  $\xi$  puede ser arbitrariamente grande, intuitivamente el intervalo en el que ambas funciones tienen soporte “*casi disjunto*” crece de forma indefinida lo cual nos permite realizar la siguiente aproximación teniendo en cuenta que  $g_1(t) = \widehat{f}$ :

$$\left\| |\widehat{f}| - |\widehat{L_\tau f}| \right\| \sim \|g_1(t)\| = \|f\|.$$

Donde la última igualdad la hemos realizado gracias a la fórmula de Plancharel que en el caso de  $\mathbb{R}^d$  es:

$$\int_{\mathbb{R}^d} |f(x)|^2 dx = \int_{\mathbb{R}^d} |\widehat{f}(\omega)|^2 d\omega. \quad (2.4)$$

Así, en vista de lo obtenido anteriormente, no es posible encontrar una constante  $c \in \mathbb{R}$  tal que la desigualdad (2.1) se cumpla para cualquier valor de  $\epsilon$ .

El razonamiento anterior no ha tenido en cuenta la hipótesis de que la función  $f$  debe tener soporte compacto, pero se ha realizado de esta manera por simplicidad en las cuentas, aunque puede adaptarse a este caso también.

Para tratar de arreglar el problema del módulo de la transformada de Fourier reemplazare-

## 2. Modelización Matemática de una Red Neuronal Convolutacional



Figura 2.1.: Como podemos ver en la imagen, para los valores  $\epsilon = 0.1$ ,  $\xi = 100$  y  $M = 5$  ambas funciones tienen soporte casi disjunto de manera que la diferencia entre ellas en el intervalo  $[-5, 5]$  coincide prácticamente con  $g_1(t)$ .

mos las ondas sinusoidales de la transformada de Fourier por funciones localizadas con un soporte mayor en altas frecuencias que tendrán un mejor rendimiento para nuestro propósito. Estas funciones se denominan **ondeletas**.

### 2.1.2. Alternativa: Las ondeletas

Las ondeletas, son pequeñas ondas estables bajo la acción de deformaciones, a diferencia de las ondas sinusoidales de Fourier. Definiremos la transformada de ondeletas y veremos que calcula, mediante convoluciones con bases de ondeletas, coeficientes estables bajo la acción de difeomorfismos.

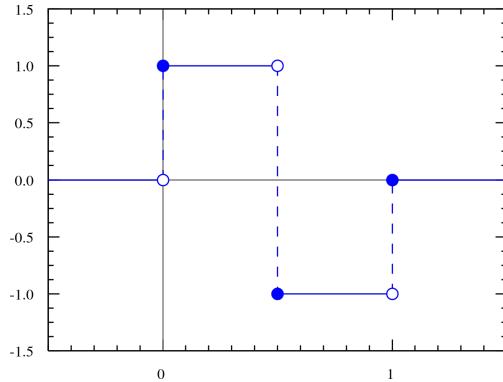
Al contrario que las bases de Fourier, las bases de ondeletas definen representaciones dispersas de señales regulares a trozos, que podrían incluir transiciones y singularidades. En las imágenes, los mayores coeficientes de las ondeletas se localizan en el entorno de las esquinas y en las texturas irregulares.

A modo de ejemplo vamos a ver la base de Haar que, aunque no sea la que utilicemos para construir nuestro propagador de dispersión, puede ayudar a entender mejor la filosofía de las ondeletas. Lo que sigue está basado en [Maloo].

Se construye a partir de la siguiente función:

$$\psi(t) = \begin{cases} 1 & 0 \leq t < 1/2 \\ -1 & 1/2 \leq t < 1 \\ 0 & \text{en otro caso,} \end{cases}$$

podemos ver una representación gráfica en la [Figura 2.2.](#)



[Figura 2.2.](#): Representación gráfica de la ondeleta de Haar.

A esta ondeleta la denominamos **ondeleta Madre**, pues a partir de ella, podemos generar la siguiente base ortonormal

$$\left\{ \psi_{j,n}(t) = \frac{1}{\sqrt{2^j}} \psi\left(\frac{t - 2^j n}{2^j}\right) \right\}_{(j,n) \in \mathbb{Z}^2}$$

del espacio  $L^2(\mathbb{R})$  de señales con energía finita.

Así, cualquier señal  $f$  de energía finita puede ser representada por los coeficientes que se obtienen mediante el producto interno en  $L^2(\mathbb{R})$  con la base anterior:

$$\langle f, \psi_{j,n} \rangle = \int_{-\infty}^{+\infty} f(t) \psi_{j,n}(t) dt$$

y puede recuperarse sumando en su base ortonormal:

$$f = \sum_{j=-\infty}^{+\infty} \sum_{n=-\infty}^{+\infty} \langle f, \psi_{j,n} \rangle \psi_{j,n}$$

Esto nos permite (igual que pasaba con el módulo de la Transformada de Fourier) trabajar en un dominio más sencillo donde procesar la información con mayor rapidez y posteriormente reconstruir la señal a partir de los coeficientes sin perder información. Algunas propiedades de la base de Haar son:

## 2. Modelización Matemática de una Red Neuronal Convolutacional

- Cada ondeleta  $\psi_{j,n}$  tiene media 0 en su soporte  $[2^j n, 2^j(n+1)]$ .
- Si  $f$  es localmente regular y el intervalo donde tiene soporte la ondeleta correspondiente es muy pequeño, debido a las propiedades de  $f$ , la función en el intervalo  $[2^j n, 2^j(n+1)]$  será prácticamente constante, lo que se traduce en que su coeficiente de ondeleta  $\langle f, \psi_{j,n} \rangle$  es prácticamente cero.
- Los mayores coeficientes se localizan en los cambios bruscos de intensidad de señal, como pueden ser los bordes, las esquinas o las texturas en las imágenes, pues en estos casos somos capaces de encontrar un elemento de la base de Haar con cuyo soporte esté en el intervalo dónde se produzca el cambio de intensidad, y por lo tanto que tenga un coeficiente de ondeletas distinto de cero.

Para el caso concreto de imágenes (ver sección 1.1 de [Maloo]), las bases de ondeletas ortonormales pueden construirse a partir de bases ortonormales en señales de una dimensión. Lo haremos a partir de tres ondeletas para capturar las variaciones horizontales, verticales y diagonales presentes en la imagen. Así, denominamos a las ondeletas usadas como  $\psi^1(x)$ ,  $\psi^2(x)$  y  $\psi^3(x)$  con  $x = (x_1, x_2) \in \mathbb{R}^2$ , dilatadas por el factor  $2^j$  y trasladadas por  $2^j n$  con  $n = (n_1, n_2) \in \mathbb{Z}^2$ , se construye una base ortonormal para el espacio  $L^2(\mathbb{R}^2)$ :

$$\left\{ \psi_{j,n}^k(x) = \frac{1}{\sqrt{2^j}} \psi^k \left( \frac{x - 2^j n}{2^j} \right) \right\}_{(j,n) \in \mathbb{Z}^2}$$

El soporte de la ondeleta  $\psi_{j,n}^k(x)$  es un cuadrado proporcional a la escala  $2^j$  como podemos ver en la Figura 2.3. Las bases de ondeletas en dos dimensiones se discretizan para definir bases ortonormales de imágenes de N píxeles.



Figura 2.3.: En el ejemplo 1) podemos ver un ejemplo de filtro generado por el soporte de una ondeleta para la detección de líneas bordes verticales, en la imagen 2) podemos ver un ejemplo de filtro generado por el soporte de una ondeleta para la detección de bordes horizontales, y en el ejemplo 3) para las diagonales. Todas tienen soporte cuadrado.

Igual que en el caso unidimensional, los coeficientes de ondeletas  $\langle f, \psi_{j,n}^k \rangle$  serán pequeños si  $f(x)$  es regular, y serán grandes cerca de los cambios bruscos de frecuencias como en los bordes o esquinas de las imágenes, como podemos ver en Figura 2.4.

Volviendo al propósito de definir el propagador de dispersión, la ondeleta madre que elijamos y la base ortogonal que forme se verán afectadas normalmente por escalados y rotaciones, por lo tanto definimos:



Figura 2.4.: Ejemplos de aplicar la base de Haar a dos imágenes. Los filtros resaltan los bordes en tres direcciones, horizontal (derecha) vertical (abajo) y en diagonal (abajo derecha). Imagen extraída de [PJDoMo6].

**Definición 2.1.3.** Una ondeleta madre escalada por un factor  $2^j$  con  $j \in \mathbb{Z}$  y rotada por  $r \in G$  siendo  $G$  el grupo finito de rotaciones, se escribe:

$$\psi_{2^j r}(x) = 2^j \psi(2^j r^{-1} x).$$

Su transformada de Fourier es  $\widehat{\psi}_{2^j r}(\omega) = \widehat{\psi}(2^{-j} r^{-1} \omega)$ .

La transformada de dispersión que usaremos tendrá una base de ondeletas generada por una ondeleta madre del tipo:

$$\psi(x) = e^{i\eta x} \Theta(x)$$

donde  $\widehat{\Theta}(x)$  es una función real con soporte en una bola de baja frecuencia en  $x = 0$ , cuyo radio es del orden de  $\pi$ .

Y como podemos ver:

$$\widehat{\psi}(\omega) = \int_{\mathbb{R}^d} e^{i\eta x} \Theta(x) e^{-i\omega x} dx = \int_{\mathbb{R}^d} \Theta(x) e^{ix(\omega - \eta)} dx = \widehat{\Theta}(\omega - \eta).$$

Por lo tanto,  $\widehat{\psi}(\omega)$  es real y con soporte en una bola de mismo radio pero centrada en  $\omega = \eta$ , que tras el escalado y rotación, queda:

$$\widehat{\psi}_\lambda(\omega) = \widehat{\Theta}(\lambda^{-1}\omega - \eta),$$

donde  $\lambda = 2^j r \in 2^{\mathbb{Z}} \times G$ .

Por lo tanto  $\widehat{\psi}_\lambda(\omega)$  tiene soporte en una bola centrada en  $\lambda^{-1}\eta$  con radio proporcional a  $|\lambda| = 2^j$ .

## 2. Modelización Matemática de una Red Neuronal Convolutional

### 2.1.3. La Transformada de Littlewood-Paley

Una vez conocemos, un poco más en profundidad, las ondeletas y su funcionamiento, pasamos a presentar la **Transformada de ondeleta de Littlewood-Paley**, que es la que emplearemos para construir el propagador de dispersión. Su expresión es la siguiente:

$$\forall x \in \mathbb{R}^d \quad W[\lambda]f(x) = f * \psi_\lambda(x) = \int f(u)\psi_\lambda(x-u)du.$$

Donde  $*$  denota la operación de convolución.

Calculamos su transformada de Fourier, para ello tendremos en cuenta el teorema de Convolución de la Transformada de Fourier:

**Teorema 2.1.4.** *Sean  $f$  y  $g$  dos funciones integrables. Si*

$$h(x) = (f * g)(x) = \int f(y)g(x-y)dy$$

entonces:

$$\widehat{h}(\omega) = \widehat{f}(\omega)\widehat{g}(\omega)$$

$y$

$$h(x) = (f * g)(x) = \int \widehat{f}(\omega)\widehat{g}(\omega)e^{-i\omega x}d\omega.$$

De esta manera, se tiene que:

$$\widehat{W[\lambda]f}(\omega) = \widehat{f}(\omega)\widehat{\psi_\lambda}(\omega) = \widehat{f}(\omega)\widehat{\psi}(\lambda^{-1}\omega).$$

Además, teniendo en cuenta la propiedad que nos dice que si la función  $f$  es real entonces su transformada coincide con el conjugado complejo  $\widehat{f}(-\omega) = \overline{\widehat{f}(\omega)}$ , podemos ver que:

- si  $\widehat{\psi}(\omega)$  y  $f$  son reales entonces  $W[-\lambda]f = \overline{W[\lambda]f}$ , utilizando la misma propiedad de antes. Además, si denotamos por  $G^+$  al cociente de  $G$  sobre  $\{-1, 1\}$ , conjunto en el cual las dos rotaciones  $r$  y  $-r$  son equivalentes, sería suficiente calcular  $W[2^j r]f$  para las rotaciones "positivas" de  $G^+$ .
- En cambio, si  $f$  fuese compleja, entonces  $W[2^j r]f$  tendría que calcularse para todo  $r \in G$ .

Podemos entender, que cuanto menor sea el factor de escala que afecta a la ondeleta, más "*comprimida*" estará esta, y viceversa. Por lo tanto podemos establecer una relación entre la frecuencia y la escala:

- A menor escala, más comprimida estará la ondeleta, los cambios en la señal se detectarán más rápidamente y las frecuencias obtenidas serán mayores. Véase la [Figura 2.5](#).
- A mayor escala, la ondeleta estará más dilatada, los cambios se detectan en menor medida (solo si son lo suficientemente bruscos) y las frecuencias obtenidas serán menores.



Figura 2.5.: Como podemos ver, en el caso de la izquierda la ondeleta (en color morado) se ve afectada por una menor escala, lo que le permitirá detectar con mayores frecuencias los cambios que se producen en la señal al convolucionar con esta a lo largo del tiempo. En cambio, en el segundo caso, se ve afectada por una escala mayor, lo que le impedirá detectar con tanta precisión los cambios que se produzcan en la señal. Imagen extraída de [Wor].

Con la transformada de Littlewood-Paley ocurre lo mismo, a una cierta escala  $2^J$  (con  $J \in \mathbb{Z}$  fijo), solo retiene las ondeletas afectadas por un factor de escala  $2^j > 2^{-J}$ . Así, podemos obtener un umbral a partir del cual la base ortonormal de ondeletas no sería capaz de identificar cambios de frecuencias. Surge la necesidad, por tanto, de promediar las bajas frecuencias que no son cubiertas por estas ondeletas en un dominio proporcional al factor  $2^J$ :

$$A_J f = f * \phi_{2^J} \text{ con } \phi_{2^J}(x) = 2^{-J} \phi(2^{-J}x).$$

Donde  $\phi$  es una función fija, de la que dependerá la transformada de ondeleta.

Así, si  $f$  fuese real, entonces la transformada de ondeleta tendría la siguiente expresión:

$$W_J f = \{A_J f, (W[\lambda]f)_{\lambda \in \Lambda_J}\}$$

Es decir, estaría formada por el promedio de todas las ondeletas de la base que no tienen soporte a la escala fijada  $2^J$ , y el conjunto de coeficientes producidos al convolucionar cada elemento de la base con  $2^j > 2^{-J}$  con la señal  $f$ . Para denotar esto indexamos por  $\Lambda_J = \{\lambda = 2^j r : r \in G^+, 2^j > 2^{-J}\}$ .

Su norma sería:

$$\|W_J f\|^2 = \|A_J f\|^2 + \sum_{\lambda \in \Lambda_J} \|W[\lambda]f\|^2. \quad (2.5)$$

Si  $J = \infty$  entonces todas las ondeletas de la base obtendrían coeficientes no nulos y por lo tanto

$$W_\infty f = \{W[\lambda]f\}_{\lambda \in \Lambda_\infty},$$

con  $\Lambda_\infty = 2^\mathbb{Z} \times G^+$ .

Su norma en este caso sería

$$\|W_\infty f\|^2 = \sum_{\lambda \in \Lambda_\infty} \|W[\lambda]f\|^2.$$

En el caso en que  $f$  sea compleja, se incluyen todas las rotaciones y obtenemos  $W_J f = \{A_J f, (W[\lambda]f)_{-\lambda, \lambda \in \Lambda_J}\}$  y  $W_\infty f = \{W[\lambda]f\}_{-\lambda, \lambda \in \Lambda_\infty}$ .

## 2. Modelización Matemática de una Red Neuronal Convolucional

La siguiente proposición da una condición estándar de Littlewood-Paley para que  $W_J$  sea unitario.

**Proposición 2.1.5.** *Para cualquier  $J \in \mathbb{Z}$  o  $J = \infty$ ,  $W_J$  es unitario en el espacio de funciones reales o complejas de  $L^2(\mathbb{R}^d)$  si y solo si para casi todo  $\omega \in \mathbb{R}^d$  se cumple:*

$$\beta \sum_{j=-\infty}^{\infty} \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2 = 1 \quad y \quad |\widehat{\phi}(\omega)|^2 = \beta \sum_{j=-\infty}^0 \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2, \quad (2.6)$$

donde  $\beta = 1$  para funciones complejas y  $\beta = \frac{1}{2}$  para funciones reales.

*Demostración.* Si  $f$  es una función compleja, tenemos que  $\beta = 1$  y vamos a demostrar que (2.6) es equivalente a :

$$\forall J \in \mathbb{Z} \quad \left| \widehat{\phi}(2^J\omega) \right|^2 + \sum_{j>-J, r \in G} \left| \widehat{\psi}(2^{-j}r^{-1}\omega) \right|^2 = 1. \quad (2.7)$$

Para ello, partimos de que si sustituimos  $\beta = 1$  en (2.6) se tiene que:

$$\sum_{j=-\infty}^{\infty} \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2 = 1 \quad y \quad |\widehat{\phi}(\omega)|^2 = \sum_{j=-\infty}^0 \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2.$$

Si ahora sumamos  $\sum_{j=0}^{\infty} \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2$  en el segundo término obtenemos:

$$|\widehat{\phi}(\omega)|^2 + \sum_{j=0}^{\infty} \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2 = 1.$$

Por otro lado si vamos a la expresión a la que queremos llegar se tiene que:

$$\begin{aligned} \forall J \in \mathbb{Z} \quad & \left| \widehat{\phi}(2^J\omega) \right|^2 + \sum_{j>-J, r \in G} \left| \widehat{\psi}(2^{-j}r^{-1}\omega) \right|^2 = 1 \iff \\ \iff \forall J \in \mathbb{Z} \quad & \left| \widehat{\phi}(2^J\omega) \right|^2 = \sum_{j=-\infty}^{-J} \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2. \end{aligned}$$

Con lo que si demostramos esto último tendríamos que (2.6) y (2.7) son equivalentes para el caso  $\beta = 1$ .

$$\begin{aligned} \left| \widehat{\phi}(2^J\omega) \right|^2 &= \sum_{j=-\infty}^0 \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}2^J\omega)|^2 \\ &= \sum_{j=-\infty}^0 \sum_{r \in G} |\widehat{\psi}(2^{J-j}r^{-1}\omega)|^2 \\ &= \sum_{j=-\infty}^{-J} \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2 \end{aligned}$$

con lo que queda demostrado que (2.6) y (2.7) son equivalentes. Teniendo en cuenta que

$\widehat{W[2^J r]} f(\omega) = \widehat{f}(\omega) \widehat{\psi}_{s/r}(\omega)$ , multiplicando (2.7) por  $|\widehat{f}(\omega)|^2$  obtenemos:

$$\forall J \in \mathbb{Z} \quad \left| \widehat{\phi}(2^J \omega) \right|^2 |\widehat{f}(\omega)|^2 + \sum_{j > -J, r \in G} \left| \widehat{f}(\omega) \right|^2 \left| \widehat{\psi}(2^{-j} r^{-1} \omega) \right|^2 = |\widehat{f}(\omega)|^2.$$

Si ahora integramos en ambos miembros en  $\mathbb{R}^d$  obtenemos:

$$\int_{\mathbb{R}^d} \left( \left| \widehat{\phi}(2^J \omega) \right|^2 |\widehat{f}(\omega)|^2 + \sum_{j > -J, r \in G} \left| \widehat{f}(\omega) \right|^2 \left| \widehat{\psi}(2^{-j} r^{-1} \omega) \right|^2 \right) d\omega = \int_{\mathbb{R}^d} |\widehat{f}(\omega)|^2 d\omega.$$

Si le aplicamos la identidad de Plancharel (2.4) se obtiene:

$$\int_{\mathbb{R}^d} \left( \left| \widehat{\phi}(2^J \omega) \right|^2 |f(\omega)|^2 + \sum_{j > -J, r \in G} |f(\omega)|^2 \left| \widehat{\psi}(2^{-j} r^{-1} \omega) \right|^2 \right) d\omega = \int_{\mathbb{R}^d} |f(\omega)|^2 d\omega.$$

Si ahora recordamos la expresión (2.5), tenemos que la expresión anterior equivale a:

$$\|A_J f\|^2 + \sum_{\lambda \in \Lambda_j} \|W[\lambda]f\|^2 = \|W_J f\|^2 = \|f\|^2,$$

que es válido para todo  $J$  y en particular también cuando  $J = \infty$ .

Recíprocamente, si tenemos que  $\|W_J f\|^2 = \|f\|^2$  entonces (2.7) se verifica para casi todo  $\omega$ . De no ser así podríamos construir una función  $f$  no nula cuya transformada de fourier  $\widehat{f}$  tuviera soporte en el dominio de  $\omega$  donde (2.7) no fuera válido, y en estos casos al aplicar la fórmula de Plancherel se verificaría que  $\|W_J f\|^2 \neq \|f\|^2$  contradiciendo la hipótesis. Y como la expresión (2.7) era equivalente a la que nos daba el teorema tenemos demostrado el resultado para el caso en que  $f$  sea compleja.

Si ahora  $f$  es real entonces  $|\widehat{f}(\omega)| = |\widehat{f}(-\omega)|$  lo que implica que  $\|W[2^J r]f\| = \|W[-2^J r]f\|$ . Por lo que  $\|W_J f\|$  permanece constante si restringimos  $r$  a  $G^+$  y multiplicando  $\psi$  por  $\sqrt{2}$  se obtiene la condición (2.6) con  $\beta = \frac{1}{2}$ .  $\square$

#### 2.1.4. Convenios para futuras secciones

Llegados a este punto, ya tenemos la transformada de ondeletas que vamos a utilizar para la construcción del PD, ahora vamos a establecer algunas características que impondremos a los distintos elementos que la componen y que usaremos de ahora en adelante:

- $\widehat{\psi}$  es una función real que satisface la condición (2.6). Lo que implica que  $\widehat{\psi}(0) = \int \psi(x) dx = 0$  y  $|\widehat{\phi}(r\omega)| = |\widehat{\phi}(\omega)| \quad \forall r \in G$ .
- $\widehat{\phi}(\omega)$  es real y simétrica, por lo que  $\phi$  también lo será y  $\phi(rx) = \phi(x) \quad \forall r \in G$ .
- Las derivadas de  $\phi$  pertenecen a  $L^1(\mathbb{R}^d)$ .

A continuación introducimos algo de notación que emplearemos de ahora en adelante:

## 2. Modelización Matemática de una Red Neuronal Convolutacional

- Se denota  $g \circ f(x) = f(gx)$  a la acción de un elemento del grupo  $g \in G$ .
- Un operador  $\mathcal{R}$  parametrizado por  $p$  es denotado por  $\mathcal{R}[p]$  y  $\mathcal{R}[\Omega] = \{\mathcal{R}[p]\}_{p \in \Omega}$ .

Un cambio de variable en la integral de la transformada de ondeleta nos muestra que si  $f$  se escala y rota,  $2^l g \circ f = f(2^l gx)$  con  $2^l g \in 2^{\mathbb{Z}} \times G$ , entonces la transformada de ondeleta se escala y rota de acuerdo a:

$$W[\lambda](2^l g \circ f) = 2^l g \circ W[2^{-l} g \lambda]f.$$

Como  $\phi$  es invariante a rotaciones en  $G$ , podemos comprobar que  $A_J$  commuta con las rotaciones de  $G$ :  $A_J(g \circ f) = g \circ A_J f \quad \forall g \in G$ .

### 2.2. El operador de dispersión sobre un camino ordenado

La transformada de Littlewood-Paley definida anteriormente es Lipschitz-continua bajo la acción de difeomorfismos porque las ondeletas son funciones regulares y localizadas. Sin embargo, todavía no es invariante a traslaciones y  $W[\lambda]f = f * \psi_\lambda$  se traslada cuando lo hace  $f$ . Así, nuestro próximo objetivo será conseguir calcular coeficientes que sean invariantes a traslaciones, que permanezcan estables bajo la acción de difeomorfismos y que retengan la información en altas frecuencias que proporcionan las ondeletas, reuniendo todas estas características tendríamos el operador que necesitamos para la construcción del PD.

Los coeficientes invariantes por traslaciones los obtendremos gracias a la acción de un operador no lineal aplicando el siguiente lema:

**Lema 2.2.1.** Si  $U[\lambda]$  es un operador definido en  $L^2(\mathbb{R}^d)$ , no necesariamente lineal pero que commuta con traslaciones, entonces  $\int_{\mathbb{R}^d} U[\lambda]f(x)dx$  es invariante a traslaciones si es finito.

*Demostración.* Sea  $f \in L^2(\mathbb{R}^d)$ ,  $c \in \mathbb{R}^d$  y  $L_c f(x) = f(x - c)$  una traslación de  $f$ , como  $U[\lambda]f$  commuta con traslaciones se tiene que:

$$\begin{aligned} U[\lambda]L_c f(x) &= U(f(x - c)) \\ &= U(f)(x - c) \\ &= L_c U[\lambda]f(x) \end{aligned}$$

Vamos a comprobar ahora que si  $\int_{\mathbb{R}^d} U[\lambda]f(x)dx$  es finito, entonces la integral es invariante a traslaciones. En otras palabras, queremos comprobar que :

$$\int_{\mathbb{R}^d} U[\lambda]L_c f(x)dx = \int_{\mathbb{R}^d} U[\lambda]f(x)dx$$

Para ello, si tenemos en cuenta la commutatividad del operador  $U[\lambda]$  se tiene que

$$\begin{aligned} \int_{\mathbb{R}^d} U[\lambda]L_c f(x)dx &= \int_{\mathbb{R}^d} U[\lambda](f(x - c))dx \\ &= \int_{\mathbb{R}^d} U[\lambda](f)(x - c)dx. \end{aligned}$$

## 2.2. El operador de dispersión sobre un camino ordenado

Y tras esto basta tener en cuenta el cambio de variable  $y = x - c$  que tiene Jacobiano  $J = 1$  y se tendría que en la expresión anterior

$$\int_{\mathbb{R}^d} U[\lambda](f)(x - c) dx = \int_{\mathbb{R}^d} U[\lambda](f)(y) dy.$$

Por lo que la integral es invariante por traslaciones.  $\square$

En nuestro caso  $W[\lambda]f = f * \psi_\lambda$  es un ejemplo trivial de este lema, pues se trata de un operador que conmuta con traslaciones y  $\int_{\mathbb{R}^d} f * \psi(x) dx = 0$  porque  $\int_{\mathbb{R}^d} \psi(x) dx = 0$ .

Esto nos enseña, que para obtener un operador invariante por traslaciones y no trivial  $U[\lambda]f$ , es necesario componer  $W[\lambda]$  con un operador extra  $M[\lambda]$  que sea no lineal, y que se conoce como “demodulación”, que transforma  $W[\lambda]f$  en una función de menor frecuencia con integral distinta de cero. Además, la elección de  $M[\lambda]$  debe preservar la Lipschitz-continuidad bajo la acción de difeomorfismos. En resumen, queremos un operador no lineal que produzca coeficientes invariantes por traslaciones no triviales y que además conserve la Lipschitz-continuidad.

Vamos a poner un ejemplo para entender mejor lo que se ha comentado anteriormente:

### 2.2.1. Ejemplo para obtener coeficientes invariantes por traslaciones

Si la **ondeleta madre** fuese  $\psi(x) = e^{i\eta x} \Theta(x)$ , entonces los elementos de la base tendrían la forma  $\psi_\lambda(x) = e^{i\lambda\eta x} \Theta_\lambda(x)$ , y por lo tanto

$$\begin{aligned} W[\lambda]f(x) &= f * \phi_\lambda(x) \\ &= f * e^{i\lambda\eta x} \Theta_\lambda(x) \\ &= e^{i\lambda\eta x} (e^{-i\lambda\eta x} f(x) * \Theta_\lambda(x)) \\ &= e^{i\lambda\eta x} (f^\lambda * \Theta_\lambda(x)), \end{aligned} \tag{2.8}$$

con  $f^\lambda(x) = e^{-i\lambda\eta x} f(x)$ .

En este caso, se podría obtener un operador invariante por traslaciones si se cancela el término de modulación  $e^{i\lambda\eta x}$  con una función  $M[\lambda]$  pertinente. Por ejemplo:

$$M[\lambda]h(x) = e^{-i\lambda\eta x} e^{-i\Phi(\widehat{h}(\lambda\eta))} h(x).$$

Dónde  $\Phi(\widehat{h}(\lambda\eta))$  es la fase compleja de  $\widehat{h}(\lambda\eta)$ . Este registro de fase no lineal garantiza que  $M[\lambda]$  conmuta con las traslaciones, ya que:

$$\begin{aligned}
\int_{\mathbb{R}^d} M[\lambda]W[\lambda]f(x)dx &= \int_{\mathbb{R}^d} e^{-i\lambda\eta}e^{-i\Phi(\widehat{W[\lambda\eta]f})} \left( e^{i\lambda\eta x} \left( e^{-i\lambda\eta x} f * \Theta_\lambda(x) \right) \right) dx \\
&= e^{-i\Phi(\widehat{f}(\lambda\eta)\widehat{\psi}_\lambda(\lambda\eta))} \int_{\mathbb{R}^d} e^{-i\lambda\eta x} f * \Theta_\lambda(x) dx \\
&= e^{-i\Phi(\widehat{f}(\lambda\eta)\widehat{\psi}_\lambda(\lambda\eta))} \int_{\mathbb{R}^d} e^{-i\lambda\eta x} f(x) dx \int_{\mathbb{R}^d} \Theta_\lambda(x) dx \\
&= e^{-i\Phi(\widehat{f}(\lambda\eta)\widehat{\psi}_\lambda(\lambda\eta))} \cdot \widehat{f}(\lambda\eta) \cdot \widehat{\Theta}_\lambda(0) \\
&= |\widehat{f}(\lambda\eta) \cdot \widehat{\Theta}_\lambda(0)|^2 \\
&= |\widehat{f}(\lambda\eta)|^2 |\widehat{\Theta}_\lambda(0)|^2 \\
&= |\widehat{f}(\lambda\eta)|^2 |\widehat{\Theta}(0)|^2
\end{aligned}$$

que como podemos ver, la integral tiene un valor no trivial y por otra parte obtenemos el módulo de la transformada que como habíamos visto en el Lema 2.1.2 era invariante por traslaciones. No obstante, no utilizaremos este operador para nuestro propósito pues además de ser complejo no verifica la invarianza bajo la acción de difeomorfismos.

### 2.2.2. El operador módulo

En nuestro caso, para preservar la Lipschitz-continuidad bajo la acción de difeomorfismos necesitamos que  $M[\lambda]$  commute con éstos y que además sea no expansiva para garantizar la estabilidad en  $L^2(\mathbb{R}^d)$ . Se puede comprobar que entonces  $M[\lambda]$  tiene que ser necesariamente un operador punto a punto [JB12], lo que significa que el operador  $M[\lambda]h(x)$  que buscamos dependería únicamente del valor de  $h$  en el punto  $x$ .

Para obtener mejores propiedades vamos a imponer también que  $\|M[\lambda]h\| = \|h\| \quad \forall h \in L^2(\mathbb{R}^d)$ , lo que implica entonces que  $|M[\lambda]h| = |h|$ , ya que:

$$\begin{aligned}
\|M[\lambda]h\| = \|h\| &\iff \left( \int_{\mathbb{R}^d} |M[\lambda]h(x)|^2 dx \right)^{\frac{1}{2}} = \left( \int_{\mathbb{R}^d} |h(x)|^2 dx \right)^{\frac{1}{2}} \\
&\iff \int_{\mathbb{R}^d} |M[\lambda]h(x)|^2 dx = \int_{\mathbb{R}^d} |h(x)|^2 dx \\
&\iff \int_{\mathbb{R}^d} |M[\lambda]h(x)|^2 - |h(x)|^2 dx = 0 \\
&\iff |M[\lambda]h(x)|^2 - |h(x)|^2 = 0 \\
&\iff |M[\lambda]h(x)| = |h(x)|.
\end{aligned}$$

Para llegar a la conclusión que sugiere el autor [Mal12], se ha supuesto que  $|M[\lambda]h(x)| \geq |h(x)| \quad \forall x \in \mathbb{R}^d$ , que junto con el hecho de que las dos funciones que se restan en el integrando son positivas, nos dan el resultado buscado.

Para satisfacer todas las restricciones, utilizaremos el operador  $M[\lambda]h = |h|$ , que además elimina todas las variaciones de fase [BM13]. Se obtiene entonces de (2.8) que este módulo transforma  $W[\lambda]f$  en una señal de menor frecuencia que la original:

$$M[\lambda]W[\lambda]f = |W[\lambda]f| = |f^\lambda * \Theta_\lambda|.$$

Vamos a visualizar con un ejemplo cómo, al interferir dos señales con este operador, la frecuencia resultante es menor que cada una de las originales.

Si

$$f(x) = \cos(\xi_1 x) + a \cos(\xi_2 x)$$

donde  $\xi_1 > 0$  y  $\xi_2 > 0$  están en la banda de frecuencia cubierta por  $\widehat{\psi}_\lambda$ , entonces al aplicar el operador módulo obtenemos:

$$|f * \psi_\lambda(x)| = 2^{-1} |\widehat{\psi}_\lambda(\xi_1) + a \widehat{\psi}_\lambda(\xi_2) e^{i(\xi_2 - \xi_1)x}|$$

que oscila entre la frecuencia de interferencias  $|\xi_2 - \xi_1|$ , que como vemos es menor que  $|\xi_1|$  y  $|\xi_2|$ .

De esta manera, por la forma en que hemos construido el operador  $U[\lambda]f$  la integración de  $\int_{\mathbb{R}^d} U[\lambda]f(x)dx = \int_{\mathbb{R}^d} |f * \psi_\lambda(x)|dx$  es invariante por traslaciones, pero elimina todas las altas frecuencias de  $|f * \psi_\lambda(x)|$ . Para recuperarlas, el PD calcula los coeficientes de ondeletas para cada  $U[\lambda]f$  que son  $\{U[\lambda]f * \psi_{\lambda'}\}_{\lambda'}$ . De nuevo, los coeficientes invariantes a traslaciones se obtienen con el módulo  $U[\lambda']U[\lambda]f = |U[\lambda]f * \psi_{\lambda'}|$  y después integrando  $\int_{\mathbb{R}^d} U[\lambda']U[\lambda]f(x)dx$ .

Veamos esto con el mismo ejemplo de antes  $f(x) = \cos(\xi_1 x) + a \cos(\xi_2 x)$  pero con  $a < 1$ . Si  $|\xi_2 - \xi_1| << |\lambda|$  con  $|\xi_2 - \xi_1|$  en el soporte de  $\widehat{\psi}_{\lambda'}$ , entonces  $U[\lambda']U[\lambda]f$  es proporcional a  $a \cdot |\psi_\lambda(\xi_1)| \cdot |\psi_{\lambda'}(|\xi_2 - \xi_1|)|$ . La segunda ondeleta  $\widehat{\psi}_{\lambda'}$  captura las interferencias creadas por el módulo, entre la frecuencia de las componentes de  $f$  y el soporte de  $\widehat{\psi}_\lambda$ .

### 2.2.3. Definición de camino de frecuencias y propiedades.

**Definición 2.2.2.** Una secuencia ordenada  $p = (\lambda_1, \lambda_2, \dots, \lambda_m)$  con  $\lambda_k \in \Lambda_\infty = 2^\mathbb{Z} \times G^+$  se denomina **camino**. Al camino vacío se le denota por  $p = \emptyset$ .

**Definición 2.2.3.** Un propagador de dispersión (PD) es un producto de operadores de la forma  $U[\lambda]f = M[\lambda]W[\lambda]f = |f * \psi_\lambda| = |\int_{\mathbb{R}^d} f(u)\psi_\lambda(x-u)du|$  para  $f \in L^2(\mathbb{R}^d)$  no comunitativos por un camino ordenado:

$$U[p]f = U[\lambda_m] \dots U[\lambda_2]U[\lambda_1],$$

$$\text{con } U[\emptyset] = Id$$

El operador  $U[p]$  está bien definido en  $L^2(\mathbb{R}^d)$  porque  $\|U[\lambda]f\| = \|f\| \leq \|\psi_\lambda\|_1 \|f\|$  para todo  $\lambda \in \Lambda_\infty$ .

El PD es por tanto una cascada de convoluciones y módulos:

$$||f * \psi_{\lambda_1} * \psi_{\lambda_2} * \dots * \psi_{\lambda_m}|$$

Cada  $U[\lambda]$  filtra la frecuencia del componente en la banda cubierta por  $\widehat{\psi}_\lambda$  y lo aplica en un espacio de frecuencias menores con la operación módulo.

## 2. Modelización Matemática de una Red Neuronal Convolutacional

A continuación vamos a probar ciertas propiedades que tienen los caminos de frecuencias tal y como los hemos descrito anteriormente. Para ello empezamos con algunas definiciones que serán de utilidad:

**Definición 2.2.4.** Escribimos la rotación y reescalado de un camino  $p$  mediante  $2^l g \in 2^{\mathbb{Z}} \times G$  como  $2^l g p = (2^l g \lambda_1, 2^l g \lambda_2, \dots, 2^l g \lambda_m)$ .

**Definición 2.2.5.** La concatenación de dos caminos  $p$  y  $p'$  se denota por

$$p + p' = (\lambda_1, \lambda_2, \dots, \lambda_m, \lambda'_1, \lambda'_2, \dots, \lambda'_{m'}).$$

En el caso particular de  $p + \lambda = (\lambda_1, \lambda_2, \dots, \lambda_m, \lambda)$

Con esta notación:

**Proposición 2.2.6.** Sean  $p, p'$  dos caminos, se tiene que :

$$U[p + p'] = U[p']U[p]$$

*Demostración.* Como  $p + p' = (\lambda_1, \lambda_2, \dots, \lambda_m, \lambda'_1, \lambda'_2, \dots, \lambda'_{m'})$  entonces siguiendo la definición de  $U[p]$  se tiene que:

$$U[p + p'] = U[\lambda'_{m'}] \dots U[\lambda'_2]U[\lambda'_1]U[\lambda_m] \dots U[\lambda_2]U[\lambda_1] = U[p']U[p]$$

□

En la [Subsección 2.1.3](#) veíamos que si  $f$  era compleja, entonces su transformada de ondeletas era  $W_{\infty} = \{W[\lambda]f\}_{\lambda, -\lambda \in \Lambda_{\infty}}$ . Pero en este caso, gracias al módulo, tras la iteración  $U[\lambda_1]f = |W[\lambda_1]f|$  sería una función real. Por tanto, las siguientes transformadas de ondeletas solo haría falta calcularlas para  $\lambda_k \in \Lambda_{\infty}$ . Por eso, los propagadores de dispersión de funciones complejas se definen sobre caminos “positivos”  $p = (\lambda_1, \lambda_2, \dots, \lambda_m)$  y caminos “negativos”  $-p = (-\lambda_1, \lambda_2, \dots, \lambda_m)$ .

Sin embargo, para simplificar cálculos, todos los resultados siguientes se harán sobre PD aplicados a funciones reales.

### 2.2.4. Construcción del operador de dispersión.

En este momento, disponemos de un operador  $U[\lambda]$  que cumple todas las condiciones deseables, por lo que en esta sección vamos a ser capaces de llegar finalmente a la modelización matemática de una CNN.

**Definición 2.2.7.** Sea  $\mathcal{P}_{\infty}$  el conjunto de todos los caminos finitos. La transformada de dispersión de  $f \in L^1(\mathbb{R}^d)$  se define para cualquier camino  $p \in \mathcal{P}_{\infty}$  como:

$$\overline{S}f(p) = \int_{\mathbb{R}^d} U[p]f(x)dx$$

El operador  $\overline{S}f(p)$  es invariante a traslaciones de  $f$ , pues el operador  $U[p]$  hemos visto que cumple las propiedades necesarias para que el valor de la integral sea finito y por lo tanto sea invariante por traslaciones, y transforma  $f \in L^1(\mathbb{R}^d)$  en una función en el camino de frecuencias variable  $p$ .

Esta definición, guarda muchas similitudes con la el módulo de la transformada de Fourier. Pero en este caso, la transformada es Lipschitz-continua bajo la acción de difeomorfismos, porque se calcula iterando en transformadas de ondeletas y módulos que, como hemos visto anteriormente, son estables.

No obstante, para problemas de clasificación, es mucho más frecuente calcular pequeños descriptores que sean invariantes por traslaciones frente a una escala predefinida  $2^J$ , manteniendo las frecuencias superiores a  $2^{-J}$ , lo que nos permite ver esta variabilidad espacial. Esto se consigue convolucionando la transformada con una ventana escalada a la frecuencia deseada, en nuestro caso  $\phi_{2^J}(x) = 2^{-J}\phi(2^{-J}x)$ .

**Definición 2.2.8.** Sea  $J \in \mathbb{Z}$  y  $\mathcal{P}_J$  el conjunto de caminos finitos  $p = (\lambda_1, \lambda_2, \dots, \lambda_m)$  con  $\lambda_k \in \Lambda_J$  y  $|\lambda_k| = 2^k > 2^{-J}$ . Una ventana de transformada de dispersión se define para todo  $p \in \mathcal{P}_J$  por

$$S_J[p]f(x) = U[p]f * \phi_{2^J}(x) = \int_{\mathbb{R}^d} U[p]f(u)\phi_{2^J}(x-u)du.$$

Donde la convolución con  $\phi_{2^J}$  localiza el propagador de dispersión en dominios proporcionales a  $2^J$ .

$$S_J[p]f(x) = ||f * \psi_{\lambda_1} | * \psi_{\lambda_2} | \dots | * \psi_{\lambda_m}| * \phi_{2^J}(x).$$

Con  $S_J[\emptyset]f = f * \phi_{2^J}$ .

Esto define una familia infinita de funciones indexadas por  $\mathcal{P}_J$ , denotada por

$$S_J[\mathcal{P}_J]f := \{S_J[p]f\}_{p \in \mathcal{P}_J}.$$

Si nos fijamos, para cada camino  $p$ ,  $S_J[p]f(x)$  es una función que actúa sobre la ventana centrada en la posición  $x$  cuyo tamaño serían intervalos de dimensión  $2^J$ . Para el caso de funciones complejas, solo tendríamos que incluir en  $\mathcal{P}_J$  los caminos negativos, y si  $f$  es real  $S_J[-p] = S_J[p]f$ . En la Sección 2.3 se comprueba que para ondeletas apropiadas,  $\|f\|^2 = \sum_{p \in \mathcal{P}_J} \|S_J[p]f\|^2$ .

Sin embargo, la energía de señal se concentra en un conjunto mucho más pequeño de caminos de frecuencias descendentes  $p = (\lambda_k)_{k \leq m}$  en el cual  $|\lambda_{k+1}| \leq |\lambda_k|$ . Esto ocurre porque el propagador  $U[\lambda]$  progresivamente lleva la energía de la señal a frecuencias cada vez menores, hasta que en cierto punto es nula.

Veamos ahora la relación que guarda este propagador de ventana con el que se definió originalmente en la Definición 2.2.7. Como  $\phi(x)$  es continua en 0, si  $f \in L^1(\mathbb{R}^d)$  se tiene que su transformada de dispersión de ventana converge punto a punto a la transformada de dispersión cuando la escala  $2^J$  tiende a  $\infty$ :

$$\begin{aligned}
 \forall x \in \mathbb{R}^d \quad & \lim_{J \rightarrow \infty} 2^J S_J[p]f(x) = \lim_{J \rightarrow \infty} 2^J U[p]f * \phi_{2^J}(x) \\
 &= \lim_{J \rightarrow \infty} 2^J \int_{\mathbb{R}^d} U[p]f(u)\phi_{2^J}(x-u)du \\
 &= \lim_{J \rightarrow \infty} 2^J \int_{\mathbb{R}^d} U[p]f(u)2^{-J}\phi(2^{-J}(x-u))du \\
 &= \int_{\mathbb{R}^d} U[p]f\phi(0)du \\
 &= \phi(0) \int_{\mathbb{R}^d} U[p]f(u)du \\
 &= \phi(0)\overline{Sf}(p).
 \end{aligned}$$

Además de la convergencia punto a punto cabe destacar que, mientras que hasta ahora se ha trabajado exclusivamente en  $L^2(\mathbb{R}^d)$ , el operador presentado se encuentra en  $L^1(\mathbb{R}^d)$ . Sin embargo, en la sección 3.2 de [Mal12], se demuestra que el operador  $\overline{Sf}(p)$ ,  $\forall f \in L^2(\mathbb{R}^d)$  y para todo camino  $p$ , pertenece a  $L^2(\mathbb{R}^d)$ . Además se da una condición suficiente que garantiza la convergencia uniforme del operador de ventana  $S_J f$  en  $\overline{Sf}$ . Sin embargo, no se consigue dar una demostración para el recíproco, a pesar de que puede apreciarse en numerosos ejemplos. Debido a la falta de tiempo y la gran complejidad técnica de la sección referenciada anteriormente, se ha optado por citarla para justificar así el motivo por el que se define en  $L^1(\mathbb{R}^d)$ .

## 2.3. Propagador de dispersión y conservación de la Norma

### 2.3.1. Proceso de dispersión del propagador.

A partir de ahora denotamos por  $S_J[\Omega] := \{S_J[p]\}_{p \in \Omega}$  y  $U[\Omega] := \{U[p]\}_{p \in \Omega}$  a la familia de operadores indexados por el conjunto de caminos  $\Omega \subset \mathcal{P}_\infty$ . Así, un dispersor de ventanas  $S_J$  puede calcularse iterando en el propagador de un paso definido anteriormente como:

$$U_J f = \{A_J f, (U[\lambda]f)_{\lambda \in \Lambda_J}\},$$

con  $A_J = f * \phi_{2^J}$  y  $U[\lambda]f = |f * \psi_\lambda|$ .

Tras calcular  $U_J f$ , aplicando de nuevo  $U_J$  a cada coeficiente  $U[\lambda]f$  se genera una familia infinita aún más grande de funciones. La descomposición se continúa iterando por recursividad aplicando  $U_J$  a cada  $U[p]f$ .

Teniendo en cuenta la Proposición 2.2.6 se tiene que  $U[\lambda]U[p] = U[p + \lambda]$ , y  $A_J U[p] = S_J[p]$ , dando lugar a:

$$U_J U[p] = \{S_J[p]f, (U[p + \lambda]f)_{\lambda \in \Lambda_J}\}.$$

Podemos por tanto establecer el comportamiento de la transformada de dispersión según la longitud  $m$  del camino que estamos empleando. Sea  $\Lambda_J^m$  el conjunto de caminos de longitud  $m$  con  $\Lambda_J^0 = \emptyset$ , entonces:

$$U_J U[\Lambda_J^m] = \{S_J[\Lambda_J^m]f, (U[\Lambda_J^{m+1}]f)_{\lambda \in \Lambda_J}\}. \quad (2.9)$$

Del hecho de que  $\mathcal{P}_J = \cup_{m \in \mathbb{N}} \Lambda_J^m$ , uno puede calcular  $S_J[\mathcal{P}_J]f$  a partir de  $f = U[\emptyset]f$  iterativamente calculando  $U_J U[\Lambda_J^m]f$  para  $m$  tendiendo a  $\infty$ , tal y como se puede ver en la imagen Figura 2.6.



Figura 2.6.: Un PD  $U_J$  aplicado a un punto de una señal  $f(x)$  calcula  $U[\lambda_1]f(x) = |f(x) * \psi_{\lambda_1}|$  y como salida a la capa  $m = 0$  se promedian los coeficientes que han dado 0 (por tener  $2^j < 2^{-J}$ ) obteniendo como salida  $S_J[\emptyset]f(x) = f(x) * \phi_{2^J}$  (como se puede ver en la flecha negra). Después se aplica de nuevo  $U_J$  a cada coeficiente  $U[\lambda_1]f(x)$  del paso anterior ( $m = 1$ )  $U[\lambda_1, \lambda_2]f(x)$  obteniendo como salida  $S_J[\lambda_1]f(x) = U[\lambda_1]f(x) * \phi_{2^J}$ . Se repite este proceso de manera recursiva para cada coeficiente  $U[p]f(x)$  y obteniendo como resultado  $S_J[p]f(x) = U[p]f(x) * \phi_{2^J}$ . Imagen extraída de [BM13].

### 2.3.2. Diferencias y similitudes con una CNN

Las operaciones de la transformada de dispersión que hemos descrito siguen la estructura general de la red neuronal convolucional introducida por LeCun [LBH15], pues se describen las redes convolucionales como una cascada de convoluciones (la transformada de ondeletas  $W[\lambda]$ ) y capas de "pooling" que usan funciones no lineales (el operador módulo  $M[\lambda]$ ), las cuales se representan en este modelo como módulos de números complejos. También se puede considerar como un operador de "pooling" la función  $\phi_{2^J}$  que se emplea para agregar coeficientes y construir un operador invariante.

Las redes neuronales convolucionales han sido empleadas con mucho éxito en tareas de reconocimiento de objetos o personas y usan normalmente kernels que no son predefinidos, sino que se aprenden mediante la técnica de back-propagation al entrenar la red. En cambio, en la modelización que se ha presentado las ondeletas que usamos son prefijadas y no se aprenden.

Siguiendo con las similitudes entre ambos modelos, si  $p$  es un camino de longitud  $m$ , entonces a  $S_J[p]f(x)$  se le denomina coeficiente de orden  $m$  a escala  $2^J$ , que en el caso de una CNN, equivaldría al tensor formado por los mapas de activación tras la convolución con el kernel de la capa  $m$  de la red.

## 2. Modelización Matemática de una Red Neuronal Convolucional

### 2.3.3. Relación con herramientas clásicas de visión por computador

Por otro lado, la modelización con los algoritmos clásicos de visión por computador como **SIFT** [Low04], se usa para calcular puntos de interés en imágenes. Así, con la ondeletas apropiadas, los coeficientes de primer orden  $S[\lambda_1]f$  serían equivalentes a los coeficientes del algoritmo. De hecho, en el artículo sobre el descriptor DAISY [TLF10] se muestra cómo esos coeficientes son aproximados por  $S_J[2^j r]f = |f * \psi_{2^j r}| * \phi_{2^j}(x)$ , donde  $\psi_{2^j r}$  es la derivada parcial de una Gaussiana calculada en imagen de escala  $2^j$  de mayor calidad, para 8 rotaciones distintas  $r$ . El filtro para promediar  $\phi_{2^j}$  es un filtro Gaussiano escalado.

### 2.3.4. Operador no expansivo.

El propagador  $U_J f = \{A_J f, (|W[\lambda]f|)_{\lambda \in \Lambda_J}\}$  es no expansivo, porque la transformada de ondas  $W_J$  es unitaria pues cumple las hipótesis de la Proposición 2.1.5 y el módulo no es expansivo en el sentido de que  $\|a| - b\| \leq |a - b|$  para cualquier  $(a, b) \in \mathbb{C}^2$ . Esto es válido tanto si  $f$  es real o compleja. Como consecuencia:

$$\begin{aligned} \|U_J f - U_J h\|^2 &= \|A_J f - A_J h\|^2 + \sum_{\lambda \in \Lambda_J} \| |W[\lambda]f| - |W[\lambda]h| \|^2 \\ &\leq \|W_J f - W_J h\|^2 \leq \|f - h\|^2 \end{aligned}$$

Al ser  $W_J$  unitaria, tomando la función nula  $h = 0$  y siguiendo el mismo razonamiento anterior, también se comprueba que  $\|U_J f\| = \|f\|$  por lo que el operador  $U_J$  preserva la norma.

Para todo conjunto de caminos  $\Omega$ , las normas de  $S_J[\Omega]f$  y  $U[\Omega]f$  son:

$$\|S_J[\Omega]f\|^2 = \sum_{p \in \Omega} \|S_J[p]f\|^2 \quad y \quad \|U[\Omega]f\|^2 = \sum_{p \in \Omega} \|U[p]f\|^2$$

Como  $S_J[\mathcal{P}_J]$  itera en  $U_J$ , que es no expansivo, la siguiente proposición prueba que  $S_J[\Omega]f$  es también no expansivo.

**Proposición 2.3.1.** *La transformada de dispersión de ventana es no expansiva:*

$$\forall (f, h) \in L^2(\mathbb{R}^d)^2 \quad \|S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]h\| \leq \|f - h\|$$

*Demostración.* Como  $U_J$  es no expansiva, partiendo de (2.9) que nos dice:

$$U_J U[\Lambda_J^m] = \{S_J[\Lambda_J^m]f, (U[\Lambda_J^{m+1}]f)_{\lambda \in \Lambda_J}\},$$

se tiene que:

$$\begin{aligned} \|U[\Lambda_J^m]f - U[\Lambda_J^m]h\|^2 &\geq \|U_J U[\Lambda_J^m]f - U_J U[\Lambda_J^m]h\|^2 \\ &= \|S_J[\Lambda_J^m]f - S_J[\Lambda_J^m]h\|^2 + \|U[\Lambda_J^{m+1}]f - U[\Lambda_J^{m+1}]h\|^2. \end{aligned}$$

Si ahora sumamos en  $m$  cuando tiende a  $\infty$  se obtiene que:

$$\sum_{m=0}^{\infty} \|U[\Lambda_J^m]f - U[\Lambda_J^m]h\|^2 \geq \sum_{m=0}^{\infty} \|S_J[\Lambda_J^m]f - S_J[\Lambda_J^m]h\|^2 + \sum_{m=0}^{\infty} \|U[\Lambda_J^{m+1}]f - U[\Lambda_J^{m+1}]h\|^2,$$

que equivale a:

$$\sum_{m=0}^{\infty} \|U[\Lambda_J^m]f - U[\Lambda_J^m]h\|^2 - \sum_{m=0}^{\infty} \|U[\Lambda_J^{m+1}]f - U[\Lambda_J^{m+1}]h\|^2 \geq \sum_{m=0}^{\infty} \|S_J[\Lambda_J^m]f - S_J[\Lambda_J^m]h\|^2$$

Si ahora nos fijamos en el lado izquierdo de la desigualdad, se cancelan todos los términos salvo  $m = 0$ , y teniendo en cuenta que  $\Lambda_J^0 = \emptyset$  queda:

$$\|U[\Lambda_J^0]f - U[\Lambda_J^0]h\|^2 = \|U[\emptyset]f - U[\emptyset]h\|^2 = \|f - h\|^2$$

Por otro lado para el miembro derecho de la desigualdad anterior, se tiene que

$$\sum_{m=0}^{\infty} \|S_J[\Lambda_J^m]f - S_J[\Lambda_J^m]h\|^2 = \|S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]h\|^2.$$

Luego hemos probado que

$$\|S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]h\|^2 \leq \|f - h\|^2$$

y por lo tanto que la transformada de dispersión de ventana es no expansiva.  $\square$

### 2.3.5. Conservación de la norma.

En la [Subsección 2.1.3](#) se obtuvo que cada coeficiente  $U[\lambda]f = |f * \psi_\lambda|$  capturaba la energía de frecuencia de  $f$  en una banda de frecuencia cubierta por  $\widehat{\psi}_\lambda$  y propagaba dicha energía a frecuencias decrecientes, este hecho lo demuestra el siguiente resultado, mostrando que toda la energía del propagador de dispersión alcanza la frecuencia mínima  $2^J$  y es atrapada por el filtro de paso bajo  $\phi_{2^J}$ . La energía propagada tiende a 0 conforme se incrementa la longitud del camino, y el teorema implica que  $\|S_J[\mathcal{P}_J]f\| = \|f\|$ . Esto se aplica también a funciones complejas en caminos negativos.

Para la demostración de la conservación de la norma necesitamos unos resultados previos:

**Lema 2.3.2.** Si  $h$  es una función tal que  $h \geq 0$  entonces  $\forall f \in L^2(\mathbb{R}^d)$ :

$$|f * \psi_\lambda * h| \geq \sup_{\eta \in \mathbb{R}^d} |f * \psi_\lambda * h_\eta| \text{ con } h_\eta = h(x)e^{i\eta x}$$

## 2. Modelización Matemática de una Red Neuronal Convolucional

*Demostración.*

$$\begin{aligned}
|f * \psi_\lambda| * h(x) &= \int \left| \int f(v) \psi_\lambda(u-v) dv \right| h(x-u) du \\
&= \int \left| \int f(v) \psi_\lambda(u-v) e^{i\eta(x-u)} h(x-u) dv \right| du \\
&\geq \left| \int \int f(v) \psi_\lambda(u-v) e^{i\eta(x-u)} h(x-u) dv du \right| = \\
&= \left| \int f(v) \int \psi_\lambda(x-v-u') h(u') e^{i\eta u'} du' dv \right| \\
&= \left| \int f(v) \psi_\lambda * h_\eta(x-v) dv \right| = |f * \psi_\lambda * h_\eta|
\end{aligned}$$

Donde se ha usado el cambio de variable  $u' = x - u$  con Jacobiano igual a 1.  $\square$

A continuación definimos el concepto de “ondeleta admisible”:

**Definición 2.3.3.** Una ondeleta de dispersión se dice que es admisible si existe  $\eta \in \mathbb{R}^d$  y una función  $\rho \geq 0$ , con  $|\hat{\rho}(\omega)| \leq |\hat{\phi}(2\omega)|$  y  $\hat{\rho}(0) = 1$ , tal que la función:

$$\widehat{\Psi}(\omega) = |\hat{\rho}(\omega - \eta)|^2 - \sum_{k=1}^{+\infty} k(1 - |\hat{\rho}(2^{-k}(\omega - \eta))|^2) \quad (2.10)$$

satisface:

$$\alpha = \inf_{1 \leq |w| \leq 2} \sum_{j=-\infty}^{\infty} \sum_{r \in G} \widehat{\Psi}(2^{-j} r^{-1} \omega) |\hat{\psi}(2^{-j} r^{-1} \omega)|^2 > 0. \quad (2.11)$$

Con esta definición en mente podemos comprobar que se da el siguiente lema que demuestra que el propagador dispersa la energía progresivamente hacia bajas frecuencias.

**Lema 2.3.4.** Si (2.11) se satisface y

$$\|f\|_w^2 = \sum_{j=0}^{\infty} \sum_{r \in G^+} j \|W[2^j r]f\|^2 < \infty$$

entonces se tiene:

$$\frac{\alpha}{2} \|U[\mathcal{P}_J]f\|^2 \geq \max(J+1, 1) \|f\|^2 + \|f\|_w^2. \quad (2.12)$$

La demostración de lema se encuentra en el apéndice A de [Mal12]. No se ha incluido en este trabajo por tratarse de una demostración muy técnica y de gran complejidad. Además, no se trata de uno de los principales resultados del trabajo, por lo que se ha optado por referenciar su demostración.

Con todos estos resultados podemos presentar el principal teorema de esta sección, que nos dará como resultado la preservación de la norma del operador de ventana:

**Teorema 2.3.5.** Si las ondeletas son admisibles, entonces para toda  $f \in L^2(\mathbb{R}^d)$  se tiene que

$$\lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 = \lim_{m \rightarrow \infty} \sum_{n=m}^{\infty} \|S_J[\Lambda_J^n]f\|^2 = 0$$

y

$$\|S_J[\mathcal{P}_J]f\| = \|f\|.$$

*Demostración.* Esta demostración tiene dos partes, la primera consistirá en demostrar que la condición (2.10) implica que  $\lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 = 0$ .

La clave de esto reside en el Lema 2.3.2, que nos da una cota inferior de  $|f * \psi_\lambda|$  convolucionada con una función positiva. Como

$$\|U[\mathcal{P}_J]f\|^2 = \sum_{m=0}^{+\infty} \|U[\Lambda_J^m]f\|^2,$$

si  $\|f\|_w < \infty$  entonces (2.12) implica que  $\lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\| = 0$ . Este resultado se extiende a  $L^2(\mathbb{R}^d)$  por densidad. Como  $\phi \in L^1(\mathbb{R}^d)$  y  $\widehat{\phi}(0) = 1$ , cualquier  $f \in L^2(\mathbb{R}^d)$  satisface  $\lim_{n \rightarrow -\infty} \|f - f_n\| = 0$ , donde  $f_n = f * \phi_{2^n}$  y  $\phi_{2^n} = 2^{-nd}\phi(2^{-n}x)$ . Se demuestra por tanto que  $\lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f_n\| = 0$  viendo que  $\|f_n\|_w < \infty$ . De hecho,

$$\begin{aligned} \|W[2^j r]f_n\|^2 &= \int |\widehat{f}(\omega)|^2 |\widehat{\phi}(2^n \omega)|^2 |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2 d\omega \\ &\leq C 2^{-2n-2j} \int |\widehat{f}(\omega)|^2 d\omega, \end{aligned}$$

porque hay un momento en que  $\psi$  desaparece, entonces  $|\widehat{\psi}(\omega)| = O(|\omega|)$ , y las derivadas de  $\phi$  están en  $L^1(\mathbb{R}^d)$  luego  $|\omega| \widehat{\phi}\omega$  están acotadas. Por lo que se tiene que  $\|f_n\|_w < \infty$ .

Como  $U[\Lambda^m]$  es no expansiva, se tiene  $\|U[\Lambda_J^m]f - U[\Lambda_J^m]f_n\| \leq \|f - f_n\|$ , por lo que

$$\|U[\Lambda_J^m]f\| \leq \|f - f_n\| + \|U[\Lambda_J^m]f_n\|.$$

Como  $\lim_{n \rightarrow -\infty} \|f - f_n\| = 0$  y  $\lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f_n\| = 0$  tenemos que

$$\lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 = 0$$

para toda  $f \in L^2(\mathbb{R}^d)$ .

**En segundo lugar** vamos a ver que las siguientes expresiones son equivalentes:

$$\lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 = 0 \iff \lim_{m \rightarrow \infty} \sum_{n=m}^{\infty} \|S_J[\Lambda_J^n]f\|^2 = 0 \iff \|S_J[\mathcal{P}_J]f\|^2 = \|f\|^2.$$

Primero probamos que

$$\lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 = 0 \iff \lim_{m \rightarrow \infty} \sum_{n=m}^{\infty} \|S_J[\Lambda_J^n]f\|^2 = 0.$$

Como  $\|U_J h\| = \|h\| \forall h \in L^2(\mathbb{R}^d)$  y  $U_J U[\Lambda_J^n]f = \{S_J[\Lambda_J^n]f, U[\Lambda_J^{n+1}]f\}$ ,

$$\|U[\Lambda_J^n]f\|^2 = \|U_J U[\Lambda_J^n]f\|^2 = \|S_J[\Lambda_J^n]f\|^2 + \|U[\Lambda_J^{n+1}]f\|^2. \quad (2.13)$$

Sumando en  $m \leq n < \infty$  se obtiene :

2. Modelización Matemática de una Red Neuronal Convolucional

$$\begin{aligned} \sum_{n=m}^{\infty} \|U[\Lambda_J^n]f\|^2 &= \sum_{n=m}^{\infty} \|S_J[\Lambda_J^n]f\|^2 + \sum_{n=m}^{\infty} \|U[\Lambda_J^{n+1}]f\|^2 \\ &\Updownarrow \\ \sum_{n=m}^{\infty} \|U[\Lambda_J^n]f\|^2 - \sum_{n=m}^{\infty} \|U[\Lambda_J^{n+1}]f\|^2 &= \sum_{n=m}^{\infty} \|S_J[\Lambda_J^n]f\|^2 \end{aligned}$$

En el término de la izquierda se anulan entre si todos los sumandos salvo  $n = m$ , luego queda:

$$\|U[\Lambda_J^m]f\|^2 = \sum_{n=m}^{\infty} \|S_J[\Lambda_J^n]f\|^2$$

Y tomando límites cuando  $m \rightarrow \infty$

$$\lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 = \lim_{m \rightarrow \infty} \sum_{n=m}^{\infty} \|S_J[\Lambda_J^n]f\|^2.$$

Por otro lado, sumando en (2.13) para  $0 \leq n < m$  se obtiene:

$$\begin{aligned} \sum_{n=0}^{m-1} \|U[\Lambda_J^n]f\|^2 &= \sum_{n=0}^{m-1} \|S_J[\Lambda_J^n]f\|^2 + \sum_{n=0}^{m-1} \|U[\Lambda_J^{n+1}]f\|^2 \\ &\Updownarrow \\ \sum_{n=0}^{m-1} \|U[\Lambda_J^n]f\|^2 - \sum_{n=0}^{m-1} \|U[\Lambda_J^{n+1}]f\|^2 &= \sum_{n=0}^{m-1} \|S_J[\Lambda_J^n]f\|^2. \end{aligned}$$

En el término de la izquierda se anulan entre si todos los sumandos salvo  $n = 0$ , y teniendo en cuenta que  $f = U[\Lambda_J^0]f$  queda:

$$\|f\|^2 = \sum_{n=0}^{m-1} \|S_J[\Lambda_J^n]f\|^2 + \|U[\Lambda_J^m]f\|^2.$$

### 2.3. Propagador de dispersión y conservación de la Norma

Si ahora tomamos límite cuando  $m \rightarrow \infty$  obtenemos:

$$\begin{aligned} \lim_{m \rightarrow \infty} \|f\|^2 &= \lim_{m \rightarrow \infty} \sum_{n=0}^{m-1} \|S_J[\Lambda_J^n]f\|^2 + \lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 \\ &\Updownarrow \\ \|f\|^2 &= \sum_{n=0}^{\infty} \|S_J[\Lambda_J^n]f\|^2 + \lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 \\ &\Updownarrow \\ \|f\|^2 &= \|S_J[\mathcal{P}_J]f\|^2 + \lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2. \end{aligned}$$

De manera que se puede apreciar claramente que

$$\|f\|^2 = \|S_J[\mathcal{P}_J]f\|^2 + \lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 = \|S_J[\mathcal{P}_J]f\|^2 \iff \lim_{m \rightarrow \infty} \|U[\Lambda_J^m]f\|^2 = 0.$$

Con lo que queda demostrado el teorema.  $\square$

#### 2.3.6. Conclusiones extraídas del teorema

La demostración muestra que el propagador dispersa la energía progresivamente a frecuencias menores. La energía de  $U[p]f$  se concentra principalmente en los caminos de frecuencia decrecientes  $p = (\lambda_k)_{k \leq m}$  para los que  $|\lambda_{k+1}| < |\lambda_k|$ .

El decrecimiento de  $\sum_{n=m}^{\infty} \|S_J[\Lambda_J^n]f\|^2$  nos sugiere que podemos descartar todos los caminos de longitud mayor que un cierto  $m > 0$ . De hecho, en tareas de tratamiento de imágenes y audio el decrecimiento numérico de  $\|S_J[\Lambda_J^n]f\|^2$  puede llegar a ser exponencial, lo que lleva a que en problemas de clasificación, por ejemplo, la longitud del camino se limite a  $m = 3$ .

El teorema además requiere de una transformada de ondeleta unitaria y admisible que satisfaga la condición de Littlewood-Paley  $\beta \sum_{(j,r) \in \mathbb{Z} \times G} |\widehat{\psi}(2^j r \omega)|^2 = 1$ .

Debe también existir una función  $\rho \geq 0$  y un  $\eta \in \mathbb{R}^d$  con  $|\widehat{\rho}(\omega)| \leq |\widehat{\phi}(2\omega)|$  tal que:

$$\sum_{(j,r) \in \mathbb{Z} \times G} |\widehat{\psi}(2^j r \omega)|^2 |\widehat{\rho}(2^j r \omega - \eta)|^2$$

sea suficientemente grande para que  $\alpha > 0$ . Esto se puede obtener como se indica en (2.3), con  $\psi(x) = e^{i\eta \cdot x} \Theta(x)$  y de hecho  $\widehat{\psi} = \widehat{\Theta}(\omega - \eta)$ , donde  $\widehat{\Theta}$  y  $\widehat{\rho}$  tienen su energía concentrada en los mismos dominios de frecuencia, que son bajos.



### 3. Invarianza por Traslaciones

Hasta ahora hemos definido el propagador de dispersión y hemos visto algunas propiedades como la conservación de la norma de la señal  $f$ . No obstante, aún queda por demostrar la invarianza por traslaciones.

#### 3.1. No expansividad del operador de ventana en conjuntos de caminos

Vamos a demostrar en primer lugar que  $\|S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]h\|$  es no expansiva cuando se incrementa  $J$ , y que de hecho converge cuando  $J \rightarrow \infty$ . Esto define una distancia límite que como veremos a continuación es invariante por traslaciones.

**Proposición 3.1.1.** *Para  $f, h \in L^2(\mathbb{R}^d)$  y  $J \in \mathbb{Z}$  se cumple*

$$\|S_{J+1}[\mathcal{P}_{J+1}]f - S_{J+1}[\mathcal{P}_{J+1}]h\| \leq \|S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]h\|. \quad (3.1)$$

*Demostración.* En primer lugar, vamos a transformar la condición que queremos demostrar en (3.1) en otra equivalente y que será más fácil de probar.

Si recordamos la definición de  $\mathcal{P}_J$ , era un conjunto de caminos finitos  $p = (\lambda_1, \dots, \lambda_m)$  tal que  $\lambda_k \in \Lambda_J$  y  $|\lambda_k| = 2^k > 2^{-J}$ . Luego todo camino  $p' \in \mathcal{P}_{J+1}$ , puede ser únicamente escrito como una extensión de un camino  $p \in \mathcal{P}_J$  donde  $p$  es el prefijo más grande de  $p'$  que pertenece a  $\mathcal{P}_J$ , y  $p' = p + q$  para algún  $q \in \mathcal{P}_{J+1}$ . De hecho, podemos definir el conjunto de todas las extensiones de  $p \in \mathcal{P}_J$  en  $\mathcal{P}_{J+1}$  como:

$$\mathcal{P}_{J+1}^p = p \cup p + 2^{-J}r + p'' \quad r \in G^+, p'' \in \mathcal{P}_{J+1}.$$

Esto define una partición disjunta de  $\mathcal{P}_{J+1} = \bigcup_{p \in \mathcal{P}_J} \mathcal{P}_{J+1}^p$ . Y deberíamos probar que dichas extensiones son no expansivas,

$$\sum_{p' \in \mathcal{P}_{J+1}^p} \|S_{J+1}[p']f - S_{J+1}[p']h\|^2 \leq \|S_J[p]f - S_J[p]h\|^2. \quad (3.2)$$

Finalmente, si nos fijamos, la condición (3.2) equivale a (3.1) sumando en todo  $p \in \mathcal{P}_J$ , luego probando (3.2) tendríamos el resultado que buscamos.

Para ello vamos a necesitar el siguiente lema:

**Lema 3.1.2.** *Para ondeletas que satisfacen la propiedad presentada en la Proposición 2.1.5, para toda función real  $f \in L^2(\mathbb{R}^d)$  y todo  $q \in \mathbb{Z}$  se verifica:*

$$\sum_{-q \geq l > -J} \sum_{r \in G^+} \|f * \psi_{2^l r}\|^2 + \|f * \phi_{2^l}\|^2 = \|f * \phi_{2^q}\|^2.$$

### 3. Invarianza por Traslaciones

*Demostración.* En primer lugar vamos a ver que de la Proposición 2.1.5 se deduce la siguiente expresión:

$$|\widehat{\phi}(2^J\omega)|^2 + \sum_{-q \geq j > -J} \sum_{r \in G^+} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2 = |\widehat{\phi}(2^q\omega)|^2.$$

Para ello, de la expresión

$$\frac{1}{2} \sum_{j=-\infty}^{\infty} \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2 = 1 \quad y \quad |\widehat{\phi}(\omega)|^2 = \frac{1}{2} \sum_{j=-\infty}^0 \sum_{r \in G} |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2,$$

se tiene de la misma forma que vimos en la demostración de la Proposición 2.1.5 que:

$$\forall J \in \mathbb{Z} \quad \left| \widehat{\phi}(2^J\omega) \right|^2 + \frac{1}{2} \sum_{j>-J, r \in G} \left| \widehat{\psi}(2^{-j}r^{-1}\omega) \right|^2 = 1.$$

Y partiendo el sumatorio obtenemos que:

$$\left| \widehat{\phi}(2^J\omega) \right|^2 + \frac{1}{2} \sum_{-q \geq j > -J, r \in G} \left| \widehat{\psi}(2^{-j}r^{-1}\omega) \right|^2 = \frac{1}{2} \sum_{j>-q, r \in G} \left| \widehat{\psi}(2^{-j}r^{-1}\omega) \right|^2 = |\widehat{\phi}(2^q\omega)|^2.$$

Ahora multiplicamos la expresión anterior por  $|\widehat{f}(\omega)|^2$ ,

$$\left| \widehat{f}(\omega) \right|^2 \left| \widehat{\phi}(2^J\omega) \right|^2 + \frac{1}{2} \sum_{-q \geq j > -J, r \in G} \left| \widehat{f}(\omega) \right|^2 \left| \widehat{\psi}(2^{-j}r^{-1}\omega) \right|^2 = \left| \widehat{f}(\omega) \right|^2 |\widehat{\phi}(2^q\omega)|^2.$$

Integramos en  $\omega$ ,

$$\begin{aligned} & \int \left| \widehat{f}(\omega) \right|^2 \left| \widehat{\phi}(2^J\omega) \right|^2 d\omega + \frac{1}{2} \sum_{-q \geq j > -J, r \in G} \int \left| \widehat{f}(\omega) \right|^2 \left| \widehat{\psi}(2^{-j}r^{-1}\omega) \right|^2 d\omega \\ &= \int \left| \widehat{f}(\omega) \right|^2 |\widehat{\phi}(2^q\omega)|^2 d\omega. \end{aligned}$$

Ahora estamos en condiciones de aplicar el Teorema 2.1.4, y nos queda que la expresión anterior equivale a:

$$\int |(f * \phi_{2^J})(x)|^2 dx + \sum_{-q \geq j > -J, r \in G} \int |(f * \psi_{2^j r})(x)|^2 dx = \int |(f * \phi_{2^q})(x)|^2 dx,$$

y teniendo en cuenta que  $f$  es real y por lo tanto que  $\|f * \psi_{2^j r}\| = \|f * \psi_{2^{j-r}}\|$  junto con la definición de la norma de  $L^2(\mathbb{R}^d)$ , se tiene

$$\sum_{-q \geq l > -J} \sum_{r \in G^+} \|f * \psi_{2^l r}\|^2 + \|f * \phi_{2^J}\|^2 = \|f * \phi_{2^q}\|^2$$

□

Vamos ahora a usar el lema anterior con la función  $g = U[p]f - U[p]h$  junto con que  $U[p]f * \phi_{2^J} = S_J[p]f$ . De esta forma se tiene:

$$\|g * \phi_{2^{J+1}}\|^2 + \sum_{r \in G^+} \|g * \psi_{2^{-J} r}\|^2 = \|g * \phi_{2^J}\|^2.$$

Así, sustituyendo el valor de  $g$  por el que hemos definido antes y aplicando la propiedad distributiva de la convolución, obtenemos:

$$\begin{aligned} \|U[p]f * \phi_{2^J} - U[p]h * \phi_{2^J}\|^2 &= \|U[p]f * \phi_{2^{J+1}} - U[p]h * \phi_{2^{J+1}}\|^2 \\ &\quad + \sum_{r \in G^+} \|U[p]f * \psi_{2^{-J} r} - U[p]h * \psi_{2^{-J} r}\|^2. \end{aligned}$$

Y esto equivale a

$$\begin{aligned} \|S_J[p]f - S_J[p]h\|^2 &= \|S_{J+1}[p]f - S_{J+1}[p]h\|^2 \\ &\quad + \sum_{r \in G^+} \|U[p]f * \psi_{2^{-J} r} - U[p]h * \psi_{2^{-J} r}\|^2. \end{aligned}$$

Aplicando ahora la propiedad de la norma de que  $\|g - h\| \geq \|g\| - \|h\|$  y como

$$|U[p]f * \psi_{2^{-J} r}| = U[p + 2^{-J} r]f,$$

se concluye que

$$\begin{aligned} \|S_J[p]f - S_J[p]h\|^2 &\geq \|S_{J+1}[p]f - S_{J+1}[p]h\|^2 \\ &\quad + \sum_{r \in G^+} \|U[p + 2^{-J} r]f - U[p + 2^{-J} r]h\|^2. \end{aligned}$$

Como  $S_{J+1}[\mathcal{P}_{J+1}]U[p + 2^{-J} r]f = \{S_{J+1}[p + 2^{-J} r + p'']\}_{p'' \in \mathcal{P}_{J+1}}$  y  $S_{J+1}[\mathcal{P}_{J+1}]f$  es no expansiva por la Proposición 2.3.1, usando la desigualdad anterior podemos escribir

$$\begin{aligned} \|S_J[p]f - S_J[p]h\|^2 &\geq \|S_{J+1}[p]f - S_{J+1}[p]h\|^2 \\ &\quad + \sum_{p'' \in \mathcal{P}_{J+1}} \sum_{r \in G^+} \|S_{J+1}[p + 2^{-J} r + p'']f - S_{J+1}[p + 2^{-J} r + p'']h\|^2, \end{aligned}$$

y en particular

### 3. Invarianza por Traslaciones

$$||S_J[p]f - S_J[p]h||^2 \geq \sum_{p'' \in \mathcal{P}_{J+1}} \sum_{r \in G^+} ||S_{J+1}[p + 2^{-J}r + p'']f - S_{J+1}[p + 2^{-J}r + p'']h||^2,$$

que demuestra (3.2).  $\square$

## 3.2. Invarianza por traslaciones

La proposición anterior nos demuestra que  $||S_J[\mathcal{P}_J] - S_J[\mathcal{P}_J]h||$  es positivo y no creciente cuando  $J$  se incrementa, y de hecho converge. Como  $S_J[\mathcal{P}_J]$  es no expansiva, el límite también lo es:

$$\forall f, h \in L^2(\mathbb{R}^d) \lim_{J \rightarrow \infty} ||S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]h|| \leq ||f - h||.$$

Para ondeletas de dispersión admisibles que satisfacen (2.11), el Teorema 2.3.5 nos demuestra que si  $||S_J[\mathcal{P}_J]f|| = ||f||$  entonces  $\lim_{J \rightarrow \infty} ||S_J[\mathcal{P}_J]f|| = ||f||$ . El siguiente teorema demuestra que el límite es invariante por traslaciones, pero para la demostración del teorema necesitaremos dos resultados auxiliares:

**Lema 3.2.1.** *Sea  $K : L^2(\mathbb{R}^d) \rightarrow L^2(\mathbb{R}^d)$  un operador tal que  $\forall f \in L^2(\mathbb{R}^d), Kf(x) = \int f(u)k(x, u)du$  verificando*

$$\int_y |k(x, u)|dx \leq C$$

$$\int_y |k(x, u)|du \leq C$$

con  $C > 0$ .

Entonces  $\|K\| \leq C$ . Donde  $\|K\|$  es la norma en  $L^2(\mathbb{R}^d)$  de  $K$ .

El lema anterior es conocido como Lema de Schur. Sin embargo, como se trata de un lema auxiliar que emplearemos para la demostración del teorema principal de la sección, no lo demostraríamos.

**Lema 3.2.2.** *Existe una constante  $C$  tal que para todo  $\tau \in \mathbb{C}^2(\mathbb{R}^d)$  con  $||\nabla \tau||_\infty \leq \frac{1}{2}$  se tiene que*

$$||L_\tau A_J f - A_J f|| \leq C ||f|| 2^{-J} ||\tau||_\infty.$$

*Demostración.* En esta prueba, vamos a necesitar el Lema 3.2.1.

El operador norma de  $k_J = L_\tau A_J - A_J$  se calcula aplicando el lema de Schur a su kernel,

$$k_J(x, u) = \phi_{sJ}(x - \tau(x) - u) - \phi_{2J}(x - u).$$

Si nos fijamos en la expresión anterior, cuando  $x = 0 = u$  se tiene que:

$$k_J(0, 0) = \phi_{2J}(0) - \phi_{2J}(0) = 0.$$

Si ahora calculamos su polinomio de Taylor de primer orden centrado en el  $(0, 0)$  se obtiene:

$$k_J = k_J(0, 0) + \int_0^1 \nabla \phi_{2J}(x - t\tau(x) - u)\tau(x).dt$$

Si ahora calculamos el módulo obtenemos que:

$$\begin{aligned} |k_J| &= |k_J(0, 0)| + \left| \int_0^1 \nabla \phi_{2J}(x - t\tau(x) - u)\tau(x).dt \right| \\ &\leq |k_J(0, 0)| + \left| \int_0^1 \nabla \phi_{2J}(x - t\tau(x) - u)\tau(x).dt \right| \\ &\leq \left| \int_0^1 \nabla \phi_{2J}(x - t\tau(x) - u)\tau(x).dt \right| \\ &\leq \int_0^1 |\nabla \phi_{2J}(x - t\tau(x) - u)\tau(x)| dt = |\tau(x)| \int_0^1 |\nabla \phi_{2J}(x - t\tau(x) - u)| dt \\ &\leq \|\tau(x)\|_\infty \int_0^1 |\nabla \phi_{2J}(x - t\tau(x) - u)| dt. \end{aligned}$$

Si ahora integramos en  $u$  y aplicamos el teorema de Fubini para intercambiar las integrales del lado derecho de la desigualdad obtenemos:

$$\int |k_J| du \leq \|\tau(x)\|_\infty \int \int_0^1 |\nabla \phi_{2J}(x - t\tau(x) - u)| dt du = \|\tau(x)\|_\infty \int_0^1 \int |\nabla \phi_{2J}(x - t\tau(x) - u)| du dt.$$

por otro lado, vamos a comprobar que

$$\nabla \phi_{2J}(x) = 2^{-dJ-J} \nabla \phi(2^{-J}x).$$

Para ello debemos recordar que  $\phi_{2J}(x) = 2^{-dJ} \phi(2^{-J}x)$  luego

$$\begin{aligned} \nabla \phi_{2J}(x) &= \nabla(2^{-dJ} \phi(2^{-J}x)) \\ &= 2^{-dJ} \nabla(\phi(2^{-J}x)). \end{aligned}$$

Si nos fijamos, debido a que  $x$  está multiplicado por  $2^{-J}$  en cada componente del vector, siempre que derivemos con respecto a alguna componente, vamos a poder sacar como factor común  $2^{-J}$  por lo tanto:

$$\nabla \phi_{2J}(x) = 2^{-dJ-J} \nabla(\phi(2^{-J}x)).$$

De esta forma, realizando un cambio de variable obtenemos:

$$\begin{aligned} \int |k_J| du &\leq \|\tau(x)\|_\infty 2^{-dJ-J} \int |\nabla \phi(2^Ju')| du' \\ &= 2^{-J} \|\tau(x)\|_\infty \|\nabla \phi\|_1. \end{aligned}$$

### 3. Invarianza por Traslaciones

Si ahora realizamos el mismo procedimiento integrando en  $x$  en vez de en  $u$  tenemos que

$$\int |k_J(x, u)| dx \leq \|\tau(x)\|_\infty \int_0^1 \int |\nabla \phi_{2J}(x - t\tau(x) - u)| dx dt$$

Ahora, aplicamos el cambio de variable  $v = x - t\tau(x)$  y calculamos su Jacobiano

$$\begin{aligned} Jv &= J(x - t\tau(x)) = J(x) - J(t\tau(x)) \\ &= Id - tJ(\tau(x)) \\ &= Id - t\nabla\tau(x). \end{aligned}$$

Vamos a buscar una cota para el determinante del Jacobiano

$$\begin{aligned} |J| &= (1 - t\tau(x))^d \\ &\geq (1 - \|\tau\|_\infty)^d \\ &\geq 2^{-d}. \end{aligned}$$

Aplicando ahora el cambio de variable a la integral

$$\begin{aligned} \int |k_J(x, u)| dx &\leq \|\tau(x)\|_\infty 2^d \int_0^1 \int |\nabla \phi_{2J}(v - u)| dv dt \\ &= 2^{-J} \|\tau\|_\infty \|\nabla\phi\|_1 2^d. \end{aligned}$$

De las dos cotas superiores obtenidas esta es la mayor, por lo que aplicamos el lema de Schur a esta y terminamos la demostración del lema

$$\|L_\tau A_J - A_J\| \leq 2^{-J+d} \|\nabla\phi\|_1 \|\tau\|_\infty. \quad \square$$

Con esto ya tenemos todas las herramientas necesarias para enunciar y demostrar el teorema central de esta sección, que nos garantiza que el operador que estamos construyendo y que modeliza una red neuronal convolucional, es invariante por traslaciones.

**Teorema 3.2.3.** *Para ondeletas de dispersión admisibles se tiene que*

$$\forall f \in L^2(\mathbb{R}^d), \forall c \in \mathbb{R}^d \quad \lim_{J \rightarrow \infty} \|S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]L_c f\| = 0.$$

*Demostración.* Fijamos  $f \in L^2(\mathbb{R}^d)$ . Teniendo en cuenta la conmutatividad  $S_J[\mathcal{P}_J]L_c f = L_c f S_J[\mathcal{P}_J]$  y la definición  $S_J[\mathcal{P}_J]f = A_J U[\mathcal{P}_J]f$ , obtenemos

$$\begin{aligned} \|S_J[\mathcal{P}_J]L_c f - S_J[\mathcal{P}_J]f\| &= \|L_c A_J U[\mathcal{P}_J]f - A_J U[\mathcal{P}_J]f\| \\ &\leq \|L_c A_J - A_J\| \|U[\mathcal{P}_J]f\|. \end{aligned}$$

Si ahora aplicamos el Lema 3.2.2 con  $\tau = c$ , se tiene que  $\|\tau\|_\infty = |c|$  y además

$$||L_c A_J - A_J|| \leq C 2^{-J} |c|.$$

Y si tenemos en cuenta esto en la expresión anterior nos da que:

$$\begin{aligned} ||S_J[\mathcal{P}_J]L_c f - S_J[\mathcal{P}_J]f|| &\leq ||L_c A_J - A_J|| ||U[\mathcal{P}_J]f|| \\ &\leq C 2^{-J} |c| ||U[\mathcal{P}_J]f|| \end{aligned}$$

Como la admisibilidad de la condición (2.11) se satisface, en el Lema 2.3.4 se demuestra en (2.12) que para  $J > 1$  se cumple

$$\frac{\alpha}{2} ||U[\mathcal{P}_J]f||^2 \leq (J+1) ||f||^2 + ||f||_w^2.$$

Y de esta expresión podemos sacar una cota superior para  $||U[\mathcal{P}_J]f||$ :

$$||U[\mathcal{P}_J]f||^2 \leq ((J+1) ||f||^2 + ||f||_w^2) 2\alpha^{-1}$$

Si  $||f||_w < \infty$  entonces elevando al cuadrado en la desigualdad anterior, tenemos

$$||S_J[\mathcal{P}_J]L_c f - S_J[\mathcal{P}_J]f||^2 \leq ((J+1) ||f||^2 + ||f||_w^2) C^2 2\alpha^{-1} 2^{-2J} |c|^2,$$

y tomando límite en ambos lados cuando  $J \rightarrow \infty$  tenemos que

$$\begin{aligned} \lim_{J \rightarrow \infty} ||S_J[\mathcal{P}_J]L_c f - S_J[\mathcal{P}_J]f||^2 &\leq \lim_{J \rightarrow \infty} ((J+1) ||f||^2 + ||f||_w^2) C^2 2\alpha^{-1} 2^{-2J} |c|^2 \\ &= 0. \end{aligned}$$

Luego  $\lim_{J \rightarrow \infty} ||S_J[\mathcal{P}_J]L_c f - S_J[\mathcal{P}_J]f|| = 0$ .

Finalmente vamos a probar ahora que el límite anterior se da  $\forall f \in L^2(\mathbb{R}^d)$ , con un argumento similar al de la prueba del Teorema 2.3.5. Cualquier  $f \in L^2(\mathbb{R}^d)$  se puede escribir como el límite de una sucesión de funciones  $\{f_n\}_{n \in \mathbb{N}}$  con  $||f_n||_w < \infty$ , y como  $S_J[\mathcal{P}_J]$  es no expansivo y  $L_c$  es unitario, usando la desigualdad triangular, deducimos que

$$||L_c S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]f|| \leq ||L_c S_J[\mathcal{P}_J]f_n - S_J[\mathcal{P}_J]f_n|| + 2||f - f_n||.$$

Haciendo tender  $n \rightarrow \infty$  se prueba que  $\lim_{J \rightarrow \infty} ||S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]L_c f|| = 0$  con lo que acaba la demostración.  $\square$



## 4. Conclusiones y trabajos futuros

En el primer capítulo, dimos una introducción a las CNN en el contexto de la clasificación de imágenes. Establecimos dos objetivos principales:

- Tratar de dar una posible modelización matemática para una CNN.
- Demostrar la propiedad básica de la invarianza por traslaciones.

En primer lugar, expusimos el problema de conseguir un operador adecuado que se pudiera aplicar de manera recursiva del mismo modo que hacen las CNN. Para ello, se llegó a la conclusión de que dicho operador debía ser **invariante por traslaciones** y **Lipschitz-continuo** bajo la acción de difeomorfismos, y vimos que debíamos rechazar el operador módulo de la transformada de Fourier como candidato, por no cumplir esta segunda propiedad.

Así, vimos que la alternativa más prometedora era la de usar una transformada de Ondeletas, en concreto la de **Littlewood-Paley**:

$$\forall x \in \mathbb{R}^d \quad W[\lambda]f(x) = f * \psi_\lambda(x) = \int f(u)\psi_\lambda(x-u)du.$$

Este operador, al contrario que el módulo de la transformada de Fourier, es Lipschitz-continuo bajo la acción de difeomorfismos, pero produce coeficientes que no son invariantes por traslaciones. Para ello se necesita la ayuda de un operador no lineal auxiliar  $M[\lambda]W[\lambda]f = |W[\lambda]f|$  y vimos, que usando el módulo, conseguíamos obtener un operador invariante por traslaciones definido en cualquier camino  $p \in \mathcal{P}_\infty$  como:

$$\bar{S}f(p) = \int_{\mathbb{R}^d} U[p]f(x)dx.$$

No obstante, en la práctica se empleará el operador de ventana:

$$S_J[p]f(x) = ||f * \psi_{\lambda_1} * \psi_{\lambda_2} * \dots * \psi_{\lambda_m}| * \phi_{2J}(x).$$

Con este operador, se consigue definir la posible modelización de una CNN, que además tiene unas propiedades deseables como la preservación de la norma o que es un operador no expansivo. Finalmente, con la modelización propuesta, se comprueba que no es expansivo en conjuntos de caminos a medida que decrece el umbral y con esta propiedad se consigue demostrar su invarianza por traslaciones.

Llegados a este punto, podemos afirmar haber conseguido todos los objetivos que nos habíamos planteado. Sin embargo, el camino para lograrlos ha sido complicado y ha estado marcado por la consecución de retos menores, que me han hecho aprender mucho sobre conceptos que se explicaron durante el grado y otros totalmente nuevos.

En primer lugar, tuve que recordar conceptos básicos del análisis de Fourier, como son el cálculo transformadas de Fourier. Así, como sus interpretaciones y aplicaciones. También he

#### *4. Conclusiones y trabajos futuros*

aprendido teoremas conocidos en el ámbito del análisis de Fourier como son la fórmula de Plancharel o el Teorema de convolución de la transformada de Fourier.

Todas estas herramientas me han permitido introducirme en el mundo de las ondeletas, que es esencial en áreas de conocimiento como el procesamiento de señales y la visión por computador. Tuve que aprender y entender el concepto de ondeleta y transformada de ondeletas, además tuve que realizar una investigación sobre el tratamiento de señales (especialmente de imágenes) por medio de estas herramientas.

Por otro lado, esta investigación me ha ayudado a comprender mejor el funcionamiento de una red neuronal convolucional y las posibles motivaciones matemáticas y propiedades subyacentes. Este tipo de estudio, de nuevo da sentido y sirve como nexo de unión a los dos grados que he cursado. Me he habituado a consultar y leer publicaciones matemáticas que trataran sobre estos temas, una destreza que considero de vital importancia y que, sin duda, será de gran utilidad para el futuro.

## **Parte II.**

**Localización de landmarks cefalométricos por  
medio de técnicas de few-shot learning**



## 5. Introducción

En los últimos años, el desarrollo de la Inteligencia Artificial [NIO2] ha permitido que tareas que antes se consideraban lentas o complicadas se puedan llevar a cabo con mayor rapidez y precisión. En cambio, hay otras que, incluso para expertos en la materia, todavía resultan complicadas de resolver de forma rápida y precisa. El presente trabajo de fin de grado (TFG) se ocupa de una de estas tareas: la localización automática de landmarks cefalométricos en fotografías para tareas de identificación humana forense.

Nos encontramos pues en el ámbito de las **ciencias forenses**. Aquellas que aplican el método científico a hechos presuntamente delictivos, con la finalidad de aportar pruebas a efectos judiciales. Dentro de este campo destacan principalmente dos disciplinas:

- La **Criminalística**: disciplina encargada del descubrimiento y verificación científica de presuntos hechos delictivos y quienes los cometen.
- La **Medicina Forense**: disciplina encargada de determinar el origen de las lesiones, las causas de muerte o la identificación de seres humanos vivos o muertos.

Dentro de la medicina forense se encuentra la **antropología forense**, que principalmente se encarga de la identificación de personas a partir de restos óseos. Y es en este ámbito donde se encuentran los procesos que dan sentido a este trabajo.

### 5.1. Descripción del problema y Motivación

Los **landmarks cefalométricos** son puntos localizados en la cara que nos permiten caracterizar la morfología de la cabeza. La correcta identificación de estos puntos es de extrema importancia en tareas como la extracción de índices de proporción antropométricos (por ejemplo, para problemas de clasificación o detección de patologías), la comparación facial forense y la identificación humana forense mediante la superposición craneofacial (SCF), en el cual se comparan imágenes de la persona difunta (imágenes ante-mortem) con una representación 3D de un cráneo candidato, con el fin de determinar si el cráneo pertenece a la persona de las imágenes o no. La correspondencia entre cara y cráneo se hace con ayuda del marcado en el cráneo de landmarks *craneométricos*. Así, cada landmark *cefalométrico* tiene un homólogo *craneométrico* con el mismo nombre. En la [Figura 5.1](#) podemos ver un esquema del proceso seguido.

El marcado de landmarks cefalométricos no es una tarea sencilla, pues depende de factores diversos a tener en cuenta como son la calidad de la imagen, la posición del sujeto o el tejido blando facial (que separa el punto craneométrico de su homólogo cefalométrico). El desplazamiento ocasionado por el tejido blando facial no es constante ni se produce siempre en la misma dirección, podemos ver un ejemplo de este hecho en la [Figura 5.2](#). Estos problemas, ocasionan que el experto forense tarde mucho en la tarea del marcado de landmarks y que el proceso sea difícilmente replicable. Por esto, pese a los avances actuales

## 5. Introducción

que se están llevando a cabo para automatizar esta tarea [HIWK15], la identificación de landmarks cefalométricos sigue realizándose a mano normalmente. En este contexto, el presente trabajo tratará de proporcionar una solución automática al proceso del marcado de landmarks cefalométricos.



Figura 5.1.: Etapas del proceso de superposición craneofacial. En primer lugar se realiza la recogida de imágenes ante-mortem (AM) y muestras post-mortem (PM) junto con el marcado de landmarks y escaneo 3D del cráneo candidato. Tras esto comienza el experto comienza el proceso iterativo en el que trata de hacer coincidir los landmarks cefalométricos y craneométricos. Finalmente se realiza una toma de decisiones en función de los resultados obtenidos. Imagen extraída de [MMi<sup>+</sup>20].

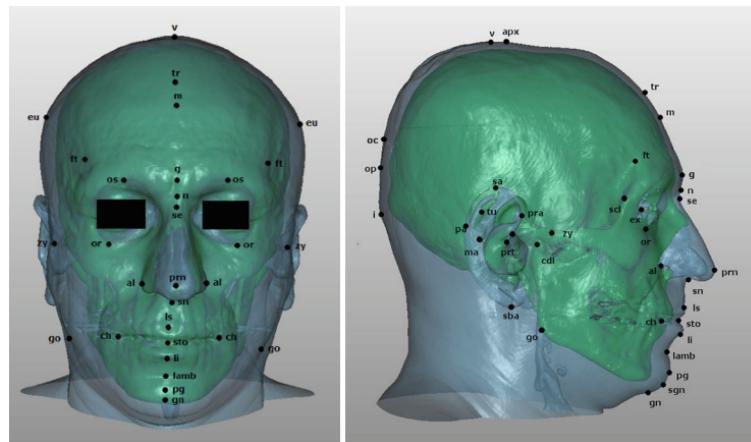


Figura 5.2.: En esta imagen podemos ver la correspondencia entre landmarks craneométricos y cefalométricos. Algunos de los landmarks que aparecen en la imagen serán estudiados en este trabajo. Imagen extraída de [DCI20].

Por otro lado, el reconocimiento de **landmarks faciales** (puntos de referencia marcados en la cara sin justificación biológica) es una tarea muy investigada actualmente, de la que existen

multitud de trabajos resueltos en su mayoría usando algoritmos de **deep learning**. Pese a no guardar relación con la morfología del cráneo, este tipo de landmarks son de gran utilidad para la identificación de personas en vídeos, el reconocimiento de gestos y posición de la cara o determinar la dirección en la que un individuo mira.

Para entrenar dichos algoritmos, se cuenta con multitud de bases de datos y ejemplos etiquetados, a diferencia que con los landmarks cefalométricos. Así, en este trabajo vamos a tratar de tomar una CNN experta en el reconocimiento de landmarks faciales en imágenes *in-the-wild* (conjunto de imágenes tomadas en situaciones no controladas de iluminación, pose o calidad), y vamos a tratar de especializarla en el marcado de **landmarks cefalométricos** aprovechando el conocimiento que ya poseen. En particular nos hemos fijado en el framework **3FabRec**[BW20]. Dicho framework cuenta con excelentes resultados en la tarea de detección de landmarks en imágenes y sobre todo nos interesa porque obtiene buenos resultados entrenando con pocas imágenes, algo que nos puede ser de utilidad teniendo en cuenta los pocos datos de los que se disponen. Así, se pretende diseñar un modelo capaz de aprender a marcar un total de 30 landmarks, que podemos ver en la [Tabla 5.1](#).

En este contexto, el presente TFG comparte el mismo problema que el TFG realizado por Guillermo Gómez en 2019, que representa una medida del estado del arte en este campo, por lo que compararemos los resultados obtenidos por su modelo con los que obtengamos en este trabajo. Además, el enfoque de los datos a usar es distinto, pues aunque partimos del mismo dataset, vamos a explorar técnicas de *few-shot learning*, es decir, trataremos de obtener un modelo competente con pocos datos de entrenamiento. Mientras que en este trabajo se amplió el conjunto de entrenamiento con un dataset extra de modelos 3D de rostros con landmarks anotados de los que se extrajeron proyecciones 2D.

### 5.1.1. Base de datos proporcionada

El conjunto de datos Forense que se proporciona para resolver el problema presenta las siguientes características:

- Contienen un total de **167 imágenes** de distintos sujetos *in-the-wild*. El número de imágenes por sujeto no se distribuye de forma equitativa, algunos solo disponen de una imagen mientras que otros tienen varias. El sujeto con mayor número de imágenes tiene siete.
- La resolución de las imágenes también varía mucho, oscilando entre los  $4350 \times 3400$  píxeles y  $168 \times 256$  píxeles.
- Hay imágenes a color y en escala de grises.
- Las imágenes se presentan en un conjunto muy variado de posiciones. Disponemos de:
  - 87 imágenes frontales.
  - 57 imágenes con rostros en posición de 3/4.
  - 23 imágenes de perfil.
- Hay hasta un total de 30 landmarks que pueden marcarse. Sin embargo, el número de landmarks en las imágenes es menor, como puede apreciarse en la [Figura 5.4](#).

Tabla 5.1.: Landmarks que se intentarán predecir.

	<b>Landmarks</b>	<b>Notación</b>
1	Menton	Me
2	Gnathion	Gn
3	Pogonion	Pg
4	Prosthion	Pr
5	Labiale Superius	Ls
6	Subnasale	Sn
7	Nasion	N
8	Glabella	G'
9	Vertex	v
10	Left Gonion	GoL
11	Right Gonion	GoR
12	Left Zygion	zyL
13	Right Zygion	zyR
14	Left Alare	alL
15	Right Alare	alR
16	Left Endocanthion	EnL
17	Right Endocanthion	EnR
18	Left Exocanthion	ExL
19	Right Exocanthion	ExR
20	Left Tragion	T'L
21	Right Tragion	T'R
22	Infradentale	Id
23	Trichion	Tr
24	Supramentale	sm
25	Left Frontotemporale	FtL
26	Right Frontotemporale	FtR
27	Left Fronozygomaticus	fzL
28	Right Fronozygomaticus	fxR
29	Left Midsurpaorbital	msoL
30	Right Midsurpaorbital	msoR

### 5.1. Descripción del problema y Motivación

- En la Figura 5.4 también podemos apreciar como la aparición de algunos landmarks es extremadamente baja, como es el caso del *prosthion* y el *Tragion* (tanto el izquierdo como el derecho). El resto de landmarks aparecen en más de la mitad de las imágenes. Los landmarks que más aparecen son los que “*a priori*” pueden aprenderse mejor. No obstante, el framework que usaremos cuenta con excelentes datos de rendimiento entrenando con pocas imágenes, por lo que los landmarks con menor aparición también pueden ser aprendidos.



Figura 5.3.: Imágenes de ejemplo del dataset proporcionado. Como puede observarse hay gran variedad de tamaños, poses, condiciones de iluminación diferentes que añaden dificultad al problema.

## 5. Introducción

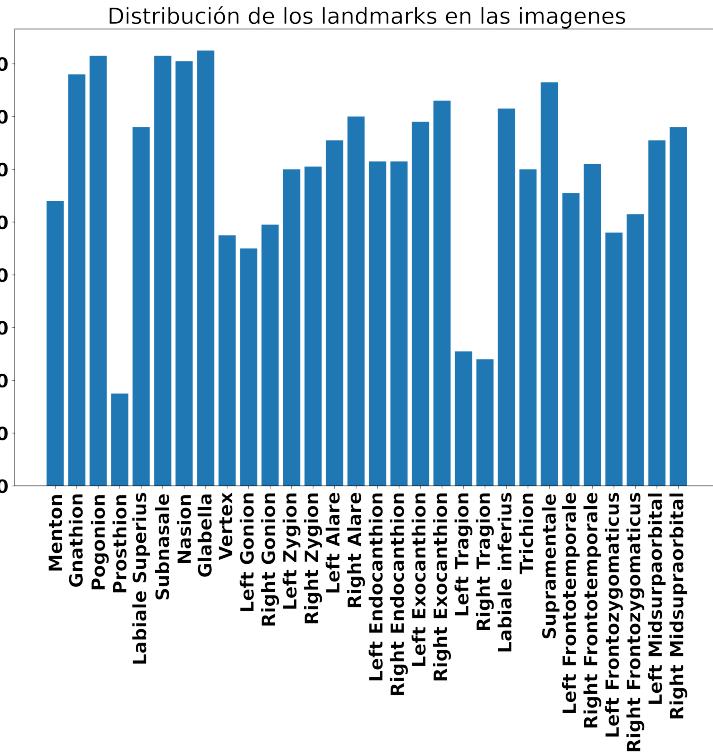


Figura 5.4.: Histograma con la aparición de cada tipo de landmark en las imágenes del dataset.

## 5.2. Requisitos mínimos del algoritmo

El algoritmo que buscamos debería tener unos requisitos mínimos:

- Debe ser un método **robusto**, en el sentido de que debe abarcar todos los posibles casos y variaciones del problema. En nuestro caso se traduce en saber lidiar con imágenes frontales, de perfil o 3/4 junto con otros factores como la calidad de la imagen, iluminación y occlusiones parciales. En todos los casos anteriores, el algoritmo debería tener un buen comportamiento.
- Capaz de operar con un **pequeño conjunto de datos**, ya que como hemos comentado anteriormente, se dispone generalmente de pocas imágenes para realizar el marcado de landmarks. Esta propiedad es deseable porque generalmente, en los problemas forenses reales, no se dispone de una amplia variedad de imágenes de la persona desaparecida, en algunos casos sólo se dispone de unas pocas imágenes y no todas van a ser frontales y con buena calidad e iluminación.
- Debe proporcionar siempre una solución lo más **correcta** posible.
- Debe ser **eficaz** y **eficiente**. Buscamos acelerar el proceso del marcado de landmarks considerablemente a la par que conseguir resolver el objetivo principal.

Remarcamos que la intención no es reemplazar al experto en su labor del marcado de landmarks sino proporcionar una ayuda para acelerar el proceso.

### 5.3. Objetivos

Los objetivos a resolver en el trabajo son los siguientes:

1. Realizar una investigación sobre el estado del arte en la localización de landmarks cefalométricos en fotografías en el ámbito de la antropología forense.
2. Realizar una investigación sobre los modelos de Autoencoders y redes adversarias existentes.
3. Explorar el dataset proporcionado para identificar errores o anomalías y pre-procesar el conjunto de datos existente, con el fin de obtener un dataset apto para el entrenamiento de la red.
4. Realizar un estudio experimental realizando diversas pruebas sobre el framework con el conjunto de datos proporcionado.

### 5.4. Planificación

La planificación del proyecto, desde un comienzo, fue pensada para llevar a cabo el desarrollo del software siguiendo un modelo en cascada, evitando la vuelta atrás entre etapas. Sin embargo, debido a que el desarrollo consistirá principalmente en una adaptación de un software ya existente, se estimó desde un principio que en esta etapa no se desarrollaría un programa de gran complejidad. Una vez adaptado el framework a nuestro objetivo, sabíamos que el software podría sufrir ligeras modificaciones en función de los resultados obtenidos durante la experimentación. Es por ello, que se optó por un modelo de ciclo de vida en cascada retroalimentado como se puede ver en [Figura 5.5](#).

Las etapas que se seguirán serán las siguientes:

1. **Análisis del problema y requisitos:** En esta tapa se profundizará sobre el contexto y la importancia del problema a resolver y se llevarán a cabo reuniones con los tutores del trabajo para aclarar los requisitos y objetivos concreto del software a desarrollar.
2. **Diseño:** En nuestro caso, en esta etapa se realizarán dos tareas:

En primer lugar se realizará un análisis de la base de datos que emplearemos así como las transformaciones que deban sufrir los datos para poder ser empleados por el software de experimentación.

En segundo lugar se diseñarán los experimentos a realizar así como las técnicas que se aplicarán, las métricas y los protocolos de validación que se usarán.

3. **Implementación:** Se implementan todas las técnicas diseñadas en la etapa anterior.
4. **Experimentación:** Se ponen en práctica todos los experimentos diseñados de forma teórica y se obtienen resultados. En función de estos se valora una vuelta atrás en el ciclo de vida del software para realizar modificaciones en el software.

## 5. Introducción

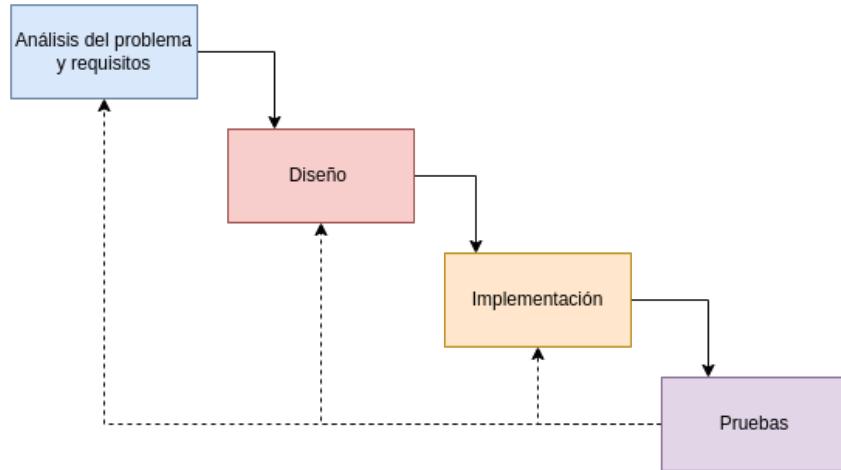


Figura 5.5.: Diseño en cascada retroalimentado empleado.

Debido a la alta carga de trabajo durante el curso, desde un comienzo, el desarrollo del trabajo se planificó pensando en su defensa para la convocatoria extraordinaria de Septiembre o para la convocatoria especial de Noviembre. Por ello, la planificación original por meses se puede observar en la [Figura 5.7](#).

Fase	Duración semanas	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto
Análisis del problema y requisitos	17								
Diseño	9								
Implementación	4								
Experimentación	5								

Figura 5.6.: Planificación original.

Dentro de la semana computan únicamente los días de Lunes a Viernes. Resalta a la vista la gran cantidad de semanas invertidas en las primeras dos etapas con respecto a las invertidas en las dos últimas. El principal motivo para repartir el tiempo de esta manera es que durante el curso se calculó que sería imposible dedicar al trabajo más de **cuatro horas por semana**. En cambio, una vez terminado el curso se previó dedicar una media de **seis horas diarias al trabajo**. Es por ello que a pesar de la gran diferencia de semanas, si miramos las horas obtenemos:

- Para la primera fase se invirtieron un total de **68 horas**.
- Para la segunda fase un total de **36 horas**.
- Para la tercera y cuarta fase se invirtieron un total de **120 horas** para cada una.
- En total el tiempo estimado de desarrollo del proyecto fue de **344 horas**.

#### 5.4. Planificación

No obstante, esta estimación resultó ser muy ajustada para completar el trabajo, lo que ocasionó que se retrasara la entrega del mismo a Noviembre, es por ello que el plan definitivo del proyecto se puede ver en la [Figura 5.7](#). Aquí se puede ver cómo desde Septiembre, a causa de los resultados obtenidos en la experimentación, se vuelve atrás en el ciclo de vida del software a la parte de análisis del problema y requisitos para consultar a los tutores con los resultados obtenidos y modificar el resto de etapas sucesivas. El tiempo invertido por día en esta segunda fase fue de **4 horas** diarias. Lo que incrementa el tiempo total invertido a:

- **148 horas** para la primera fase.
- Para la segunda fase un total de **76 horas**.
- Para la tercera y cuarta fase se invirtieron un total de **160 horas** para cada una.
- En total el tiempo estimado de desarrollo del proyecto fue de **544 horas**.

Fase	Duración semanas	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre
Análisis del problema y requisitos	22											
Diseño	11											
Implementación	6											
Experimentación	8											

Figura 5.7.: Planificación final.

Como consecuencia, en un principio se preveía usar un total de **344** horas para finalizar el trabajo, pero finalmente hicieron falta **544** horas, de esta forma, si se supone que un Investigador en una empresa de base tecnológica es de **35€/hora**, a partir de los resultados obtenidos, el trabajo tenía un presupuesto inicial de **12.040 €** que finalmente fue ampliado a **19.040 €**.



## 6. Fundamentos Teóricos y Métodos

En esta sección se introducirán los conceptos teóricos más importantes en los que se sustenta el trabajo y sus resultados. Para ello se ha recurrido al conocimiento adquirido en asignaturas como **Visión por Computador** o **Aprendizaje Automático**, así como diversos artículos que se citarán dónde sea conveniente.

### 6.1. Aprendizaje Automático

Actualmente, la **Inteligencia Artificial** (IA) es una rama de la informática de importancia creciente que pretende dotar a los ordenadores de una manera de razonar o solucionar problemas de forma inteligente. En este contexto, la IA ha explorado diversos métodos para conseguir este propósito, lo que dan lugar a un amplio árbol de investigación dónde podemos destacar el estudio de Metaheurísticas, la Ingeniería del Conocimiento y el **Aprendizaje Automático** (AA).

Los métodos que empleamos en este trabajo pertenecen a la rama del AA, y por lo tanto es importante comenzar definiendo este concepto. Para ello, disponemos de diversas definiciones proporcionadas por distintos autores.

La primera y más clásica nos la proporciona Arthur Samuel en 1959, en la cual define el AA como **el área de conocimiento que da a los ordenadores la capacidad de aprender sin ser programados explícitamente**. Esta definición es poco precisa, pero nos proporciona una perspectiva global de lo que se pretende: dotar a los ordenadores de la capacidad de “aprender” a resolver un determinad problema a partir de una base de datos de entrenamiento.

Una definición más reciente de Tom Mitchell (1998) nos dice que: **Un programa de ordenador se dice que aprende de la experiencia E en una tarea T y alguna medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E.** Esta segunda definición, a diferencia de la anterior, nos permite identificar los elementos necesarios para poder resolver un problema mediante técnicas de AA.

Así, los elementos necesarios serían una problema (T) que queremos resolver con ayuda de un ordenador, una experiencia (E) en esa tarea, que generalmente es una base de datos asociada, y una medida de rendimiento (P) que mide el rendimiento del algoritmo en la resolución del problema y que generalmente se asocia con una función objetivo que se pretende minimizar/maximizar.

Tradicionalmente, los algoritmos de AA se dividen en dos conjuntos según la naturaleza de los datos con que se entrena. De esta forma tenemos los siguientes conjuntos:

- Aprendizaje Supervisado.
- Aprendizaje no Supervisado.

Además, en los últimos años han aparecido otras técnicas como el Aprendizaje por Refugio que están siendo muy usadas, pero no vamos a profundizar en ellas pues no son necesarias para el trabajo que nos ocupa.

## 6. Fundamentos Teóricos y Métodos

### 6.1.1. Aprendizaje Supervisado

Los algoritmos de AA que se emplean en este conjunto se caracterizan porque disponen de una base de datos **etiquetados** de manera que para cada dato  $x$  conocemos su etiqueta asociada  $y$ , y el objetivo sería tratar de conocer la función  $f$  que los relaciona, de manera que  $f(x) = y$ .

#### 6.1.1.1. Regresión

En los problemas de regresión se pretende obtener la función  $f$  que asocia correctamente a cada dato su etiqueta:

$$f(x) = y \text{ con } x \in \mathbb{R}^m \text{ } y \in \mathbb{R}^n$$

Generalmente, obtener la función  $f$  de manera exacta es muy complicado, por lo que se pretende aproximar mediante una función  $f'$ , perteneciente generalmente a una familia de funciones parametrizadas, que elegimos y que entrenaremos a partir de los datos etiquetados que se nos proporcionan. Volviendo a la definición de Tom Mitchell, en este tipo de problemas la asociación con los elementos presentes en la definición sería:

- T= regresión (aproximar  $f$ )
- E= El conjunto de datos  $X$  etiquetados que se proporcionan para entrenar el modelo  $f'$ .
- P= función de coste asociada (generalmente se emplea el error cuadrático medio) que nos mide lo “bien” que nuestra función  $f'$  aproxima a  $f$ .

A modo de ejemplo, vamos a formalizar un ejemplo concreto en el cual intentamos predecir  $f$  mediante un modelo lineal, la función  $f'$  tendría la siguiente forma:

$$f'(x) = w^T x \text{ } x, w \in \mathbb{R}^m$$

Dónde  $w$  sería el conjunto de parámetros que se pretenden ajustar para aproximar lo mejor posible  $f$ . Además, disponemos de un conjunto de  $N$  datos

$$X = \{x_1, x_2, \dots, x_N\} \text{ } x_i \in \mathbb{R}^m$$

y de etiquetas

$$Y = \{y_1, y_2, \dots, y_N\} \text{ } y_i \in \mathbb{R}^n$$

Además, pongamos por ejemplo que usamos como función de coste  $J$  el error cuadrático medio, un error muy empleado en este tipo de aproximaciones:

$$J(\alpha) = \frac{1}{N} \sum_{i=1}^N (f'(x_i) - f(x_i))^2 = \frac{1}{N} \sum_{i=1}^N (y'_i - y_i)^2$$

donde  $y'_i$  es la etiqueta predicha por  $f'$  para  $x_i$ .

Con todos estos datos, nuestro objetivo sería encontrar el vector de pesos  $w$  que minimice la función de coste  $J$  y para ello utilizamos los datos de entrenamiento  $X$ .

### 6.1.1.2. Clasificación

Por otro lado tenemos los problemas de **clasificación**, en los datos se encuentran agrupados en clases y se pretende clasificar cada dato de entrada en la clase correcta. Los casos más sencillos de este problema son los de **clasificación binaria**, en ellos se pretende predecir la pertenencia o no a una determinada clase de un conjunto de datos que se codifica como 0 y 1.

En este tipo de problemas se pretende buscar una manera de definir lo mejor posible la frontera entre los diferentes tipos de datos que queremos separar. Algunos algoritmos de los más empleados en esta son los siguientes:

- **K-Nearest Neighbours(K-NN)**: En este algoritmo asociamos a cada dato la etiqueta del conjunto al que pertenece de acuerdo a los  $K$  (con  $K \in \mathbb{N}$ ) vecinos más cercanos.
- **Máquina de vector de soporte(SVM)**: Se trata de buscar el hiperplano que tenga la mayor distancia (margen) con los puntos más cercanos a él de cada conjunto.
- **Redes Neuronales**, que explicaremos en detalle en futuras secciones y destacando el *perceptrón multicapa* (MLP).

### 6.1.1.3. Gradiente Descendente

En esta sección hemos hablado de qué es el AA y cómo se formalizan sus problemas para poder resolverlos. Sin embargo, no hemos hablado de ningún algoritmo que se use en la minimización de la función de coste. Es por ello que vamos a explicar el algoritmo clásicamente se utiliza para esta tarea, el **Gradiente Descendente**.

El Gradiente descendente es un algoritmo clásico que persigue la idea intuitiva de que el gradiente de una función siempre “*apunta*” hacia el máximo de esta, por lo que seguir la dirección contraria a este nos llevará al mínimo de la función. Más formalmente, si recuperamos la notación del apartado anterior tendríamos:

La función objetivo es:

$$f(x) = y \quad x \in \mathbb{R}^m \quad y \in \mathbb{R}^n$$

La función con la que vamos a intentar aproximar la función objetivo es:

$$f'(x, w) = w^T x = y \quad x \in \mathbb{R}^m \quad y \in \mathbb{R}^n \quad w \in \mathbb{R}^d$$

La función de coste sería  $J(w)$  que de alguna manera mide la distancia entre  $f$  y  $f'$  y que para poder aplicar el método debe ser derivable. Algunas funciones de coste usuales son:

- La función **L2** (también conocida como error cuadrático medio):

$$J(w) = \frac{1}{N} \sum_{i=1}^N (f(x_i, w) - f'(x_i))^2$$

- La función **L1** (también conocida como error absoluto medio):

$$J(w) = \frac{1}{N} \sum_{i=1}^N |f(x_i, w) - f'(x_i)|$$

## 6. Fundamentos Teóricos y Métodos

No obstante, la función de coste puede variar mucho de un problema a otro, y en ocasiones no tiene por qué ser una de las anteriores, puede ser combinación lineal de varias funciones distintas o bien una función única para el problema en cuestión.

Una vez hemos formalizado el problema, el algoritmo **Gradiente Descendente** consiste en:

- Se inicializa el vector de pesos  $w$  de acuerdo a un criterio.
- En cada paso  $i$  del entrenamiento, el vector de pesos del siguiente paso  $i + 1$  se calculan los nuevos pesos usados de acuerdo a la siguiente relación:

$$w_{i+1} = w_i - \eta \nabla J(w)$$

Dónde  $\eta$  es un factor conocido como **learning rate**(lr) que mide el “*tamaño*” del paso que en cada iteración damos en búsqueda del mínimo. Y dónde  $\nabla J(w)$  es el valor del gradiente de la función de coste para el vector de pesos  $w$ .

Idealmente, con este método se encuentra un mínimo global de la función de coste en el caso en que esta sea convexa. En caso de no serlo podría caer en un mínimo local en su lugar.

Por otro lado, cabe destacar la importancia de una buena elección del **learning rate**. Si este es demasiado pequeño puede ocasionar una lenta convergencia al mínimo, y por lo tanto que se realice un gran número de iteraciones. Por otro lado, un valor excesivamente grande de este puede impedir la convergencia, pues los saltos serían tan grandes en la dirección del mínimo local o global que podría llegar a “*pasar por encima*” de este siempre. Por lo tanto una técnica habitual aunque costosa de este algoritmo consiste en usar un learning rate adaptativo, que sea mayor en las primeras iteraciones y que vaya disminuyendo conforme se incremente el número de iteraciones (pues se entiende que se estará cerca del mínimo). Podemos ver reflejado esto en la [Figura 6.1](#).

### 6.1.1.4. Gradiente Descendente Estocástico

El algoritmo descrito anteriormente tiene el problema de ser costoso computacionalmente, debido a que en cada iteración se debe calcular la función de coste para todos los ejemplos del conjunto de entrenamiento  $X$ . Es por ello, que suele emplearse en su lugar una versión modificada y que sigue dando buenos resultados que consiste en actualizar los pesos en base a unos pocos ejemplos del conjunto de entrenamiento  $X$  que se conoce como “*minibatch*”.

### 6.1.2. Aprendizaje no Supervisado

Los algoritmos de Aprendizaje no Supervisado se caracterizan porque los datos que se proporcionan no están etiquetados, y no se busca una salida concreta, sino que se pretende analizar y extraer características de nuestro conjunto de datos. Así, por ejemplo, tareas que pueden resolverse con esta técnica pueden ser la agrupación de clientes de cierta compañía en distintas clases según sus características.

### 6.1.3. Aprendizaje Automático en este Trabajo

En nuestro problema, el framework del que disponemos resuelve un problema de aprendizaje supervisado y no supervisado, pues intenta predecir un conjunto de landmarks para una

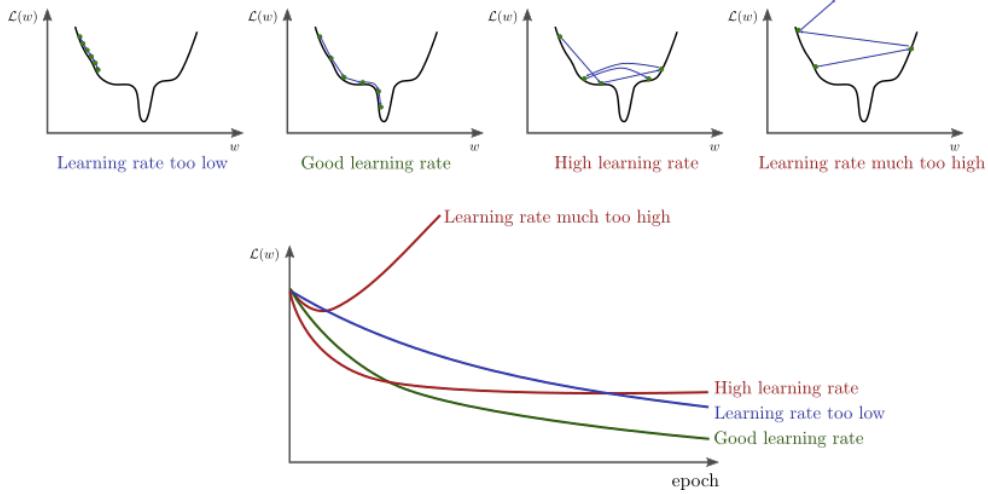


Figura 6.1.: Pretendemos alcanzar el mínimo de la función representada en las imágenes de la primera fila. Podemos ver el impacto de la elección del learning rate, si es muy pequeño, harán falta muchas iteraciones hasta la convergencia (primera imagen), si tiene un tamaño adecuado, converge rápidamente a la solución (segunda imagen), si es grande, puede quedar “atrapado” el algoritmo (tercera imagen), y si es demasiado grande podría incluso diverger. Imagen extraída de [Fei17].

cierta imagen de entrada, un problema típico de aprendizaje supervisado, en este caso de regresión dónde pretendemos a partir de la imagen de entrada conocer la función que nos proporciona la salida correcta (la imagen con los landmarks marcados correctamente).

Por otro lado, tiene una etapa de entrenamiento previa al problema de los landmarks en la cual mediante conjuntos de datos de imágenes sin etiquetar de rostros humanos, se pretende reconstruir imágenes preservando al máximo posible la estructura de la cara. Esto, como podemos ver, es un problema típico de aprendizaje no supervisado, porque no se busca obtener una etiqueta para cada imagen, sino analizar la estructura de los distintos elementos de los datos de entrada para ser capaces de reconstruirlos preservando su estructura.

Un ejemplo clásico de un problema similar es *EigenFaces*, utilizado en el reconocimiento de rostros. El objetivo era extraer los vectores propios de la matriz de covarianza del conjunto de rostros de la base de datos.

En este trabajo, sin embargo, sólo nos centraremos en la resolución del problema de **regresión** en el cual trataremos de predecir un conjunto de landmarks **cefalométricos**, y por lo tanto distintos a los que la red sabe predecir, a partir de un pequeño conjunto de imágenes etiquetadas pertenecientes a un conjunto de datos forense.

#### 6.1.4. Visión por Computador

La **Visión por computador** es un área de conocimiento en el que se unen diversas disciplinas como la IA o el AA para un propósito común, que es el procesado de imágenes por medio de un ordenador con la finalidad de que la máquina pueda llegar a extraer información relativa a estas del mismo modo en que lo haría un ser humano [Ros88].

## 6. Fundamentos Teóricos y Métodos

Problemas clásicos de la visión por computador son el reconocimiento de objetos o personas en imágenes, la segmentación o la clasificación. Así pues, podemos ver la relación directa que hay entre nuestro objetivo y esta disciplina, pues los frameworks que usaremos tendrán por objetivo extraer información de imágenes de rostros de personas para posteriormente tratar de identificar en ellos con el mayor grado de decisión posible una serie de landmarks cefalométricos que el sistema ha aprendido a base de unos ejemplos etiquetados (AA).

Finalmente, en los últimos años esta rama ha experimentado un fuerte impulso en la comunidad científica debido al actual desarrollo del **Deep Learning** y su principal herramienta: las **redes convolucionales profundas**, que explicaremos en detalle en la siguiente sección. Éstas, han permitido crear programas que obtienen un gran rendimiento en el tratamiento de imágenes.

### 6.1.5. Deep Learning

Como ya se ha mencionado anteriormente, la IA se encuentra muy desarrollada actualmente y es capaz de resolver problemas que tradicionalmente eran muy complicados para ser resueltos por un humano. Sin embargo, e irónicamente, algunas de las tareas más fáciles para los seres humanos como son el reconocimiento del habla o la identificación de objetos en imágenes han suponen un verdadero reto para un ordenador, y no ha sido hasta los últimos años con el nacimiento del **Deep Learning** que se han empezado a obtener resultados satisfactorios en este campo.

Por lo tanto, los algoritmos del Deep Learning se caracterizan por resolver estos problemas a partir de representaciones del mismo que se expresan en términos de otras más simples. De esta manera, se pueden construir conceptos difíciles a partir de otros más sencillos. Este grafo puede ser profundo en ocasiones, por ello se le conoce como Deep Learning.

#### 6.1.5.1. Redes Neuronales

La arquitectura básica de los modelos de Deep Learning viene descrita por las **redes neuronales**. Es por ello que vamos a profundizar un poco en esta estructura y partiendo de un ejemplo clásico como es el **Perceptrón multicapa**(MLP). Para esta sección vamos a seguir el capítulo 6 de [GBC16].

Si recordamos de secciones pasadas, las redes neuronales tienen por objetivo aproximar una función desconocida  $f(x) = y$  que asocia a cada entrada  $x$  una salida  $y$  a partir de una función  $f'(x; W) = \hat{y}$  (dónde  $\hat{y}$  es la etiqueta predicha para la entrada  $x$ ) que depende de unos parámetros  $W$ , de manera que el objetivo es aprender los valores de  $W$  que mejor aproximan la función objetivo  $f$ .

Los algoritmos empleados por las redes se denominan **feedforward** porque la información fluye desde la entrada  $x$  a través de los cálculos intermedios que definen  $f'$  hasta finalmente dar una salida  $\hat{y}$ . Debido a la representación gráfica de estos algoritmos, se les conoce como **redes**, pues se representan como una composición de distintas funciones en cadena de manera que se van aplicando sucesivamente sobre la entrada  $x$  hasta la salida.

La **profundidad** de la red vendría dada por la cantidad de **capas ocultas** que esta tiene.

Por otro lado, cada capa contiene un número determinado de **neuronas**, que se relacionan con las de la capa siguiente y las de la capa anterior mediante combinaciones lineales. Sin embargo, estas combinaciones lineales no son suficientes para que la red pueda aproximar



Figura 6.2.: Red neuronal con una capa oculta. Formalmente podría describirse como  $f'(x) = (f_2(f_1(x)))$ , donde  $f_2$  hace referencia a la transformación de los datos de entrada a la capa oculta y  $f_3$  de la capa oculta a la de salida.

funciones objetivo  $f$  que sean no-lineales, para ello se emplean las llamadas **funciones de activación**. Se tratan de un conjunto de funciones no-lineales entre las que destacan las siguientes [SSA17]:

- **Función Sísmoide.** Transforma los valores de entrada a un valor entre 0 y 1.

$$\text{Sísmoide}(x) = \frac{1}{1 + e^{-x}}$$

- **Función Tangente Hiperbólica.** Es similar a la función sísmoide, pero es simétrica respecto al origen.

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **Función ReLU.** Es una de las más usadas en redes neuronales por ser de las más eficientes, ya que permite que no se activen todas las neuronas a la vez. Aquellas cuya salida tras la combinación lineal con los pesos de la capa correspondiente sea o no serán activadas.

$$\text{ReLU}(x) = \max(0, x)$$

Existe el problema de que en algunos casos, el gradiente de la función sea 0 debido a que los pesos no se actualicen durante el proceso que explicaremos más adelante de **backpropagation**.

- **Función Leaky ReLU.** Es una versión mejorada de la función anterior, ya que para los casos en los que la función anterior valía 0, ahora se expresan como una componente lineal de la entrada  $x$  muy pequeña, de esta manera se resuelve el problema de que el gradiente de la función sea 0.

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{si } x \leq 0 \\ ax & \text{si } 0 < x \end{cases}$$

Dónde  $a$  es un valor muy próximo a 0.

## 6. Fundamentos Teóricos y Métodos

- **Función SoftMax.** Es una combinación de múltiples funciones sigmoide. Mientras que la función Sigmoide se usa en problemas de clasificación binaria, la función SoftMax permite que se pueda realizar clasificación multiclas, ya que transforma un vector de entrada K-dimensional de valores reales en un vector K-dimensional de elementos entre 0 y 1, de manera que la componente más próxima a 1 podría entenderse como la clase a la que pertenecería el elemento en un problema de clasificación multiclas.

$$\sigma(x)_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad j = 1, \dots, K$$

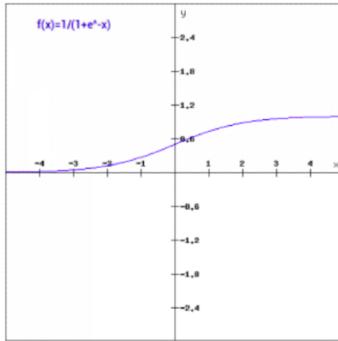


Figure 4. Sigmoid Function

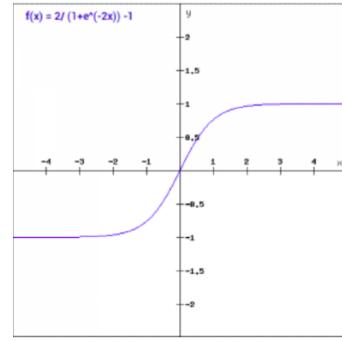


Figure 5. Tanh Function

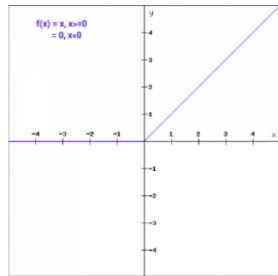


Figure 6. ReLU Activation Function plot

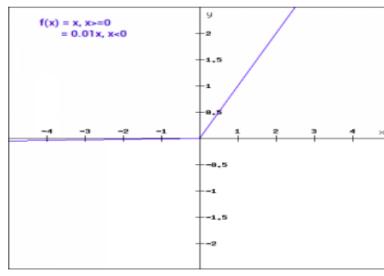


Figure 7. Plot of Leaky ReLU function

Figura 6.3.: Representación gráfica de las funciones de activación más empleadas. La imagen ha sido extraída de [SSA17].

No existe una manera de elegir la mejor función para cada caso, pero de forma experimental se ha podido comprobar que la función ReLU en general da buenos resultados y si hubiera demasiadas neuronas muertas en la red podría cambiarse por la Leaky ReLU.

De esta manera, en la capa  $i$  de la red, la función  $f_i$  suele tener la siguiente expresión:

$$f_i(x) = \gamma(w_i^T x)$$

Dónde  $\gamma$  representa una función cualquiera del conjunto de funciones de activación que hemos descrito antes y  $w_i$  el vector de pesos correspondiente a la capa  $i$  de la red. De esta manera se consigue aproximar funciones no lineales.

### 6.1.5.2. Backpropagation

Todas las funciones de las capas intermedias de la red son derivables, por lo que se podría calcular de forma explícita su derivada en cada caso, lo que ocurre es que este proceso es costoso computacionalmente. En lugar de esto se aplica la técnica de **Backpropagation**.

Para entender este proceso correctamente se debe introducir primero el concepto de **Grafo Computacional**, que no es otra cosa que representar una función mediante un grafo. Como por ejemplo [Figura 6.4](#).

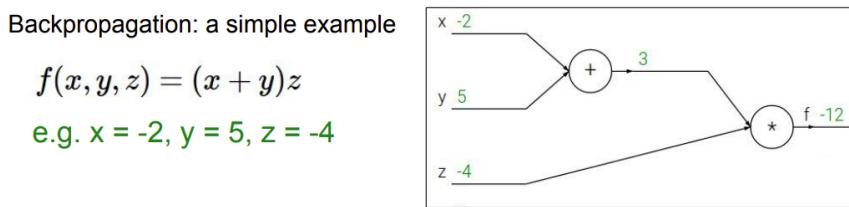


Figura 6.4.: Ejemplo de Grafo computacional junto con la salida para una entrada concreta  $x = -2, y = 5, z = -4$ . La imagen ha sido extraída del curso [Fei17].

La idea del algoritmo de Backpropagation es ir calculando la derivada en cada nodo del grafo computacional mediante la aplicación de la regla de la cadena, de manera que si  $f(x, y, z) = g(x, y)h(z)$  si quisieramos calcular  $\frac{\partial f}{\partial x}$  aplicando la regla de la cadena tendríamos que:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \frac{\partial g}{\partial x}$$

Podemos entender mejor el proceso resolviendo el ejemplo de la [Figura 6.4](#).

Como vemos en [Figura 6.5](#), se produce un flujo desde la salida hacia la entrada en la que se van calculando los gradientes para cada parámetro. Una vez calculado el gradiente se procede a actualizar los pesos usando por ejemplo el algoritmo de **gradiente descendente**.

## 6.2. Redes Neuronales Convolucionales

Las **Redes Neuronales Convolucionales** (CNN, por el inglés *Convolutional Neural Networks*) son la principal herramienta del Deep Learning y están inspiradas en las redes neuronales que hemos explicado anteriormente, con la salvedad de que ahora en vez de tener como entrada un vector de  $\mathbb{R}^d$  ahora pueden entrar a la red volúmenes de datos en varias dimensiones, como por ejemplo imágenes que se representan como  $\mathbb{R}^{w \times h \times c}$ .

## 6. Fundamentos Teóricos y Métodos

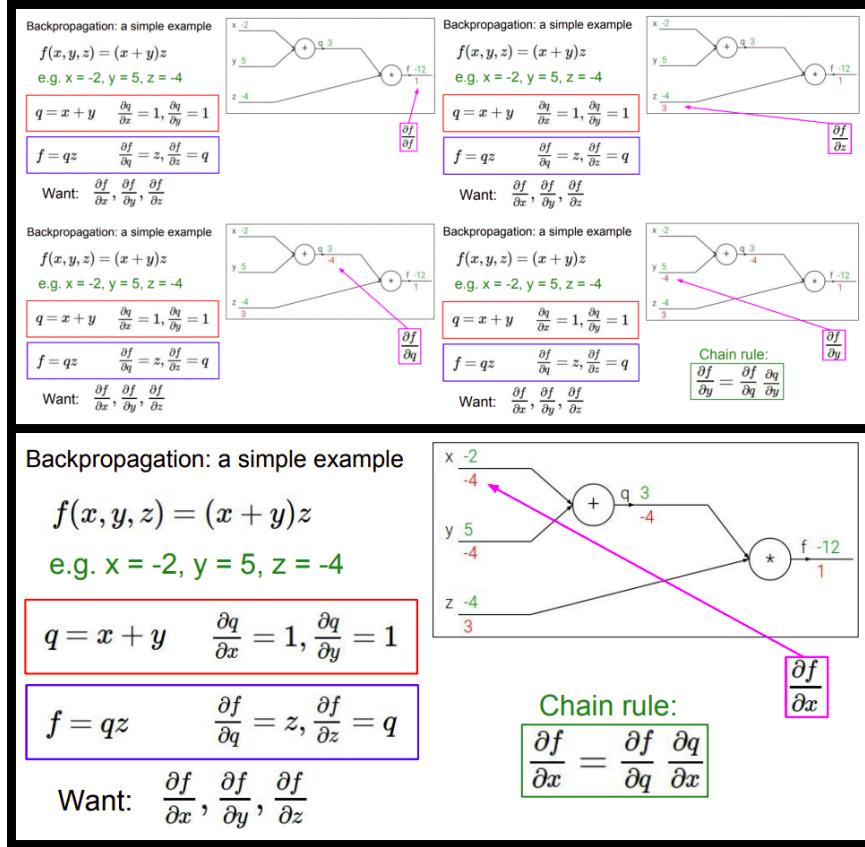


Figura 6.5.: Como podemos ver en la imagen superior, en primer lugar se renombra la salida de la operación  $x + y$  por  $q$ , de manera que  $f = qz$ . Tras esto se empiezan a calcular las derivadas parciales correspondientes desde el final hacia la entrada, aplicando cuando sea necesario la regla de la cadena hasta obtener la derivada de cada nodo en la imagen de abajo. Las imágenes han sido extraídas de [Fei17]

La principal motivación para la creación de estas redes es el concepto de **conectividad local**, pues cuando se trabaja con entradas de grandes dimensiones como son las imágenes, resulta impráctico conectar cada neurona con todas las de la capa anterior (como suele ocurrir en las redes neuronales clásicas que hemos visto anteriormente). Es por ello que se decide conectar cada neurona con una región local del volumen de entrada. Por lo que vamos a emplear filtros que siempre van a tener la misma profundidad que el volumen de entrada y que vamos a ir desplazando a lo largo de  $\mathbb{R}^{w \times h \times c}$ . A continuación, presentamos la operación que realizaremos este filtro y cuyo resultado alimenta la siguiente capa de la red.

Se denominan **Convolucionales** debido a que la principal operación matemática que realiza en cada filtro es la convolución. La operación de convolución entre dos funciones de variable real viene descrita por la siguiente expresión:

$$f(t) = \int g(a)h(t-a)da = (g * h)(t)$$

Dónde  $t$  generalmente representa el paso del tiempo y  $g, h$  son dos funciones de variable real.

Sin embargo, la expresión anterior es la definición en el caso continuo, por lo que al trabajar en un ordenador, el tiempo y las funciones que empleamos deben ser discretizado, por ello consideraremos que  $t$  toma solo valores enteros. De esta forma, definimos la **convolución discreta** en una dimensión como:

$$f(t) = (g * h)(t) = \sum_{a=-\infty}^{\infty} g(a)h(t-a)$$

Sin embargo, en nuestro caso aplicaremos las CNN a imágenes, que se discretizan como matrices 2D, por lo que necesitamos extender la definición anterior al caso de dos dimensiones:

$$f(n, m) = (g * h)(n, m) = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} g(i, j)h(n-i, m-j)$$

La convolución tiene las siguientes propiedades:

- **Commutatividad:**

$$f * g = g * f$$

- **Asociatividad:**

$$f * (g * h) = (f * g) * h$$

- **Distributividad:**

$$f * (g + h) = (f * g) + (f * h)$$

Con esto tenemos todos los ingredientes para empezar a describir la estructura de una CNN, que generalmente se compone de tres tipos de capas: **capas convolucionales**, **capas de Pooling** y **capas totalmente conectadas**.

### 6.2.1. Capa Convolucional

Se trata de la capa esencial de una CNN y consiste en desplazar un filtro por todo el volumen de entrada realizando la operación de convolución discreta en 2D. Como dijimos antes, la profundidad del filtro que empleamos para esto coincide con la del volumen de entrada siempre. Vamos a discutir ahora qué volumen de salida genera cada capa convolucional que viene definido por tres parámetros: **depth**, **stride** y **padding**

- El parámetro **depth** se corresponde con el número de filtros que queremos aplicar al volumen de entrada, pues se pretende que cada uno de ellos aprenda algo distinto sobre este. Como cada filtro produce un volumen  $m \times n$  de profundidad 1 conocido como **mapa de activación**, este parámetro nos genera tantos mapas de activación como indique.
- El parámetro **stride** indica el paso con el que vamos desplazando cada filtro sobre el volumen de entrada, así un stride de 1 indica que el filtro se desplaza de uno en uno por los píxeles, y un stride de 2 indicaría que se desplaza de dos en dos. Cuanto mayor sea el valor de este parámetro, menor será la dimensional del mapa de activación.
- El parámetro **padding** indica en cuantas filas y columnas se amplía el volumen de entrada antes de aplicar los filtros. Este parámetro se utiliza para controlar la dimensión

## 6. Fundamentos Teóricos y Métodos

de salida de los mapas de activación pues la convolución es una operación que reduce la dimensionalidad siempre, a menos que el filtro aplicado sea de  $1 \times 1$ .

Con todo esto en mente, la relación que determina el volumen de salida de una capa convolucional para un volumen de entrada de dimensión  $W \times W \times d$  y empleando filtros de dimensión  $F \times F \times d$  con un padding  $P$ , un stride  $S$  sería:

$$\text{Output} = (W - F + 2P)/S + 1$$

Generalmente la salida de cada capa convolucional supone la entrada a una función de activación como las que se describieron en [Subsección 6.1.5.1](#). Normalmente se utiliza la función ReLU, aunque también es común el uso de Leaky ReLU.

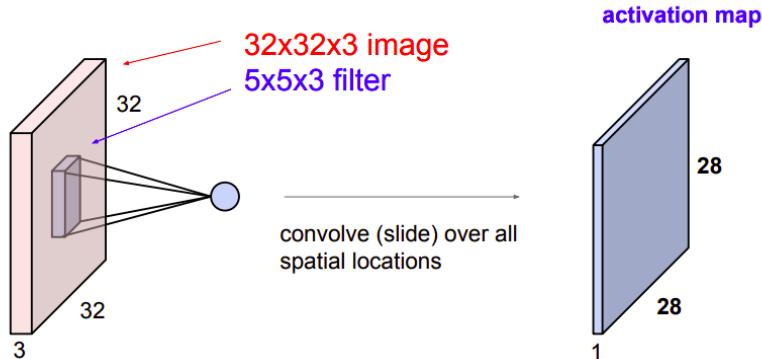


Figura 6.6.: Ejemplo de cálculo de mapa de activación en una capa convolucional para un determinado volumen de entrada. El parámetro depth nos dice la cantidad de mapas de activación generamos para el volumen de entrada, o dicho de otro modo, el número de filtros que aplicamos. La imagen ha sido extraída de [Fei17]

### 6.2.2. Capa de Pooling

En las CNN es normal insertar de vez en cuando capas de Pooling. Esta capa reduce la dimensión del volumen de entrada y actúa independientemente del volumen de la profundidad que tenga el volumen de entrada. Generalmente se emplean filtros de dimensión  $2 \times 2$  con un stride de 2 que reduce a la mitad la dimensión del volumen de entrada y mantiene la profundidad.

El tipo de operación que se realiza con el filtro  $2 \times 2$  ha sido objeto de estudio en los últimos años, y se han probado las siguientes funciones:

- **Max Pooling.** Consiste en tomar el máximo de los cuatro elementos que ve el filtro del volumen de entrada.
- **Average Pooling.** Consiste en realizar un promedio de los elementos que ve el filtro.

Generalmente, el **max pooling** parece tener un mayor rendimiento en la práctica, pero esto aún es objeto de estudio.

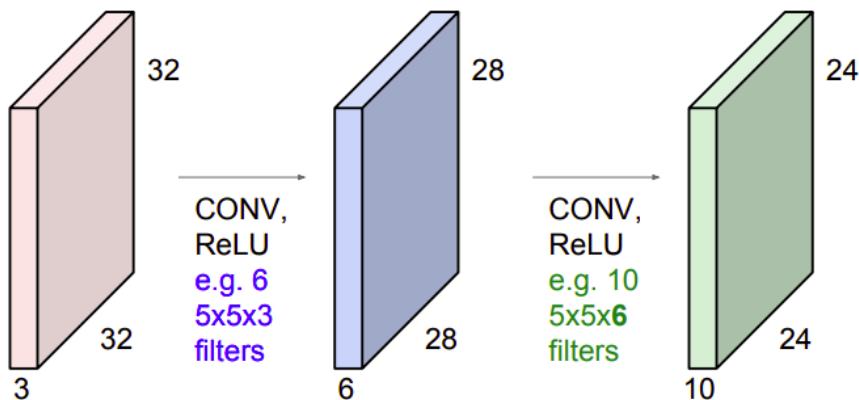


Figura 6.7.: Sucesión de varias capas convolucionales + ReLU que describen la estructura básica de una CNN. Cabe destacar como la profundidad de los filtros es siempre la misma que la del volumen de entrada, de acuerdo a lo que hemos dicho anteriormente. La imagen han sido extraída de [Fei17]

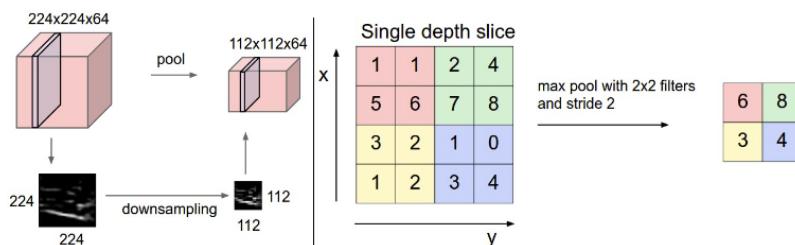


Figura 6.8.: Ejemplo de capa de pooling usando la operación del máximo. La imagen han sido extraída de [Fei17]

### 6.2.3. Capa Totalmente Conectada (Fully Connected)

Generalmente al final de las CNN suele encontrarse una capa totalmente conectada (FC) que tiene la misma estructura que una red neuronal clásica con una o dos capas ocultas generalmente, y por lo tanto está creada para un vector de entrada de una dimensión concreta.

Como hemos visto en las capas anteriores, una CNN actúa de manera independiente de las dimensiones del volumen de entrada a la red salvo en las capas totalmente conectadas, es por ello que estas capas son las que normalmente determinan el volumen de entrada a la red para que esta funcione correctamente.

### 6.2.4. Batch Normalization

Actualmente, es muy común en las CNN usar **Batch Normalization** que consiste en normalizar los datos de entrada a cada capa convolucional con la intención de hacer que las operaciones convolucionales sean independientes unas de otras, es decir, que la distribución de los datos de entrada a una capa no dependa de los parámetros aprendidos por la capa anterior. Además este procedimiento ayuda a prevenir el overfitting de la red, ayudando a la

## 6. Fundamentos Teóricos y Métodos

regularización.

### 6.2.5. Optimizador Adam

El proceso de aprendizaje en las CNN se realiza de manera similar al caso de las redes neuronales clásicas, mediante la técnica de Backpropagation. Sin embargo el optimizador Gradiente Descendente que hemos presentado en secciones anteriores tiene un único learning rate para todos los pesos de la red y se mantiene fijo durante todo el entrenamiento. Es por ello que en su lugar generalmente se usa el optimizador Adam que permite establecer un learning rate distinto para cada parámetro y que además es adaptativo.

### 6.2.6. Proceso de entrenamiento de una CNN

Una vez aclarados todos los conceptos previos estamos en condiciones de comprender el proceso clásico de entrenamiento de una CNN en el que nos basaremos para entrenar nuestro modelo en secciones posteriores:

Generalmente se entranan las imágenes por conjuntos denominados *mini-batches*, los cuales:

- En una primera instancia se propagan hacia adelante por toda la red calculando las activaciones y errores de salida en lo que se conoce como **forward pass**.
- Vamos de la salida a la entrada calculando los gradientes de cada unidad con lo que se conoce como **backward pass** aplicando el algoritmo de backpropagation.
- Se actualizan los pesos en base al gradiente calculado en el paso anterior y según el optimizador que se esté empleando (Adam, gradiente descendente, etc...)

### 6.2.7. Evolución de las CNN

Una vez hemos presentado las CNN y sus principales componentes vamos a dar un breve repaso por la historia y evolución de su arquitectura [Gup20].

#### 6.2.7.1. LeNet-5

La primera arquitectura conocida que forma parte de las CNN fue desarrollada por LeCun [LBBH98] para el reconocimiento de dígitos manuscritos, a dicha red la llamaron **LeNet-5** y fue la que sirvió de inspiración para el desarrollo de las posteriores redes.

Algunas de sus características eran:

- Su arquitectura consiste en:

*INPUT → CONV → AVG POOL → CONV → AVG POOL → FC → FC → OUTPUT.*

- Las imágenes de entrada eran de dimensión  $32 \times 32$  y pertenecían a una base de datos MNIST de dígitos manuscritos.
- Tiene alrededor de unos 60000 parámetros para entrenar.
- Las dimensiones de la imagen de entrada descienden en cada paso mientras que la profundidad del tensor aumenta hasta llegar a las capas FC.

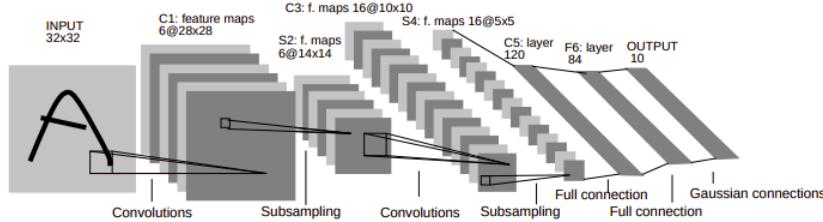


Figura 6.9.: Arquitectura de la red LeNet. Como podemos observar tiene capas convolucionales, capas de average pooling y capas totalmente conectadas al final. Imagen extraída de [LBBH98].

#### 6.2.7.2. AlexNet

Tras el éxito de LeNet comenzaron a desarrollarse nuevas arquitecturas basadas en CNN para problemas de reconocimiento de objetos en imágenes, aunque a pesar del buen rendimiento que tenían, resultaban muy costosas de entrenar para grandes volúmenes de datos o en imágenes de alta resolución. De esta manera surgió la red AlexNet [KSH12].

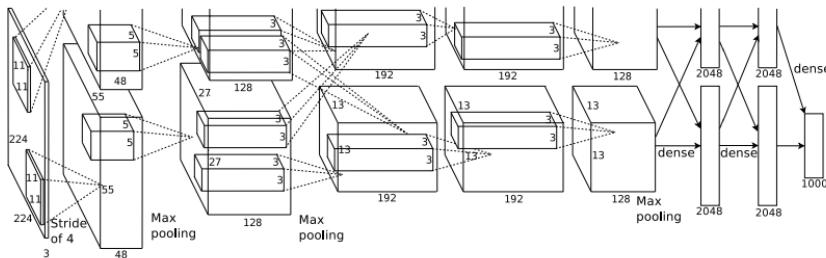


Figura 6.10.: Arquitectura de la red AlexNet. Destaca que parece estar “partida” en dos mitades, esto es porque por primera vez se usaron las GPUs para entrenar la red de manera que una GPU realizaba la parte superior de la arquitectura y otra la parte inferior. Imagen extraída de [KSH12].

Destacan las siguientes propiedades:

- Tiene un total de ocho capas sin contar las de pooling, cinco convolucionales y tres FC.
- Emplea ReLU como función de activación en lugar de tanh pues se demuestra que acelera el proceso de entrenamiento  $\times 6$  y obtiene un error del 25% en el dataset CIFAR-10.
- Se emplearon múltiples GPUs para el entrenamiento dividiendo las neuronas en dos conjuntos. Esto aceleró aún más el entrenamiento.
- Emplean la técnica de Max Pooling en lugar de la de Average Pooling de LeNet.
- Tienen 60 millones de parámetros que aprender, de manera que para evitar el Overfitting se usó la técnica de **dropout**, que consistía en “apagar” neuronas en cada iteración

## 6. Fundamentos Teóricos y Métodos

del entrenamiento de acuerdo a una probabilidad del 50 %. También se usó **data augmentation** que explicaremos en [Subsección 6.4.1](#).

- El modelo ganó la competición ImageNet en 2012 con una diferencia en precisión del 11 % con respecto al algoritmo que quedó en segundo lugar.

### 6.2.7.3. GoogLeNet

La arquitectura de GoogLeNet está basada en un módulo que se denomina **Inception** creado con la intención de reducir el coste computacional de las CNNs. Para ello se propone que en vez de construir una red muy profunda se realicen simultáneamente múltiples convoluciones en una sola capa. Con este modelo aparecen los primeros filtros de convolución  $1 \times 1$  encargados de reducir la profundidad del volumen de entrada manteniendo las dimensiones.

Con este modelo, se permitía realizar simultáneamente diversas operaciones sobre un volumen de entrada y finalmente se concatenaban los mapas de activación.

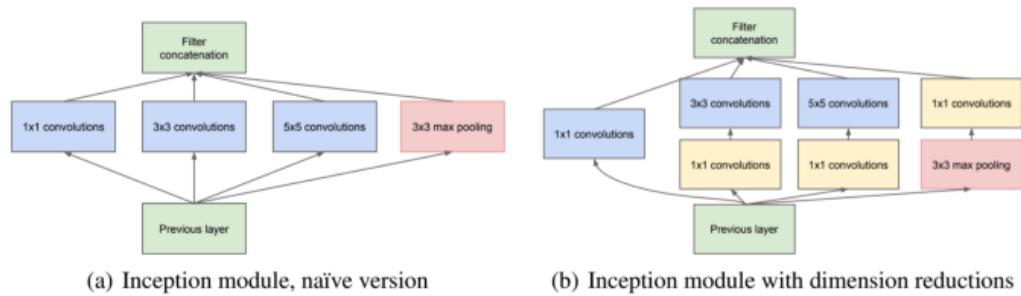


Figura 6.11.: Ejemplos de módulos Inception. Imágenes extraídas de [Fei17].

Con este nuevo módulo se desarrolló en 2015 la red GoogLeNet [SLJ<sup>+</sup>15], que consiguió ganar el concurso ImageNet con empate técnico con la red VGG-19. En vez de usar capas FC usaron la técnica de **global average pooling** mediante la cual reducían la dimensión de los tensores con convoluciones  $1 \times 1$ .

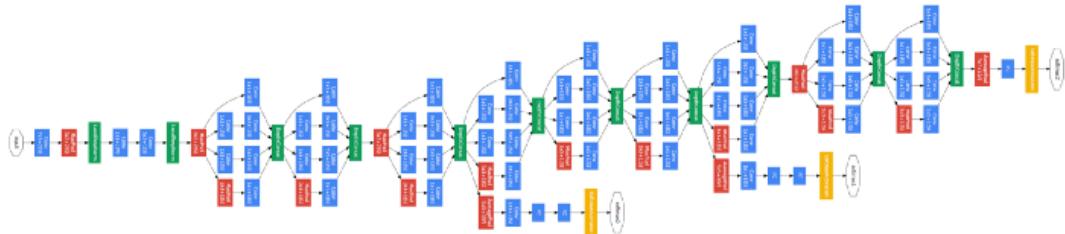


Figura 6.12.: Arquitectura de GoogLeNet usando módulos Inception. Imagen extraída de [Fei17]

Destacan las siguientes propiedades:

- Tiene un total de 27 capas de profundidad, por lo que aumenta notablemente la profundidad con respecto a sus antecesores.

- El uso de convoluciones  $1 \times 1$  con 128 filtros ayudan a reducir la dimensionalidad en lugar de usar capas FC.
- Contiene una capa FC con 1024 unidades y que alimenta una ReLU.
- Tiene una capa lineal con Softmax para clasificación multiclas.

#### 6.2.7.4. VGG-16

Supuso un gran salto en rendimiento con respecto a los modelos anteriores. Los principales cambios con modelos anteriores como AlexNet es que dejaron de usar filtros de grandes dimensiones en las primeras etapas de la red para pasar a usar solo filtros  $3 \times 3$ . Esto ahorraba costes y permitía hacer una red más profunda [SZ14].

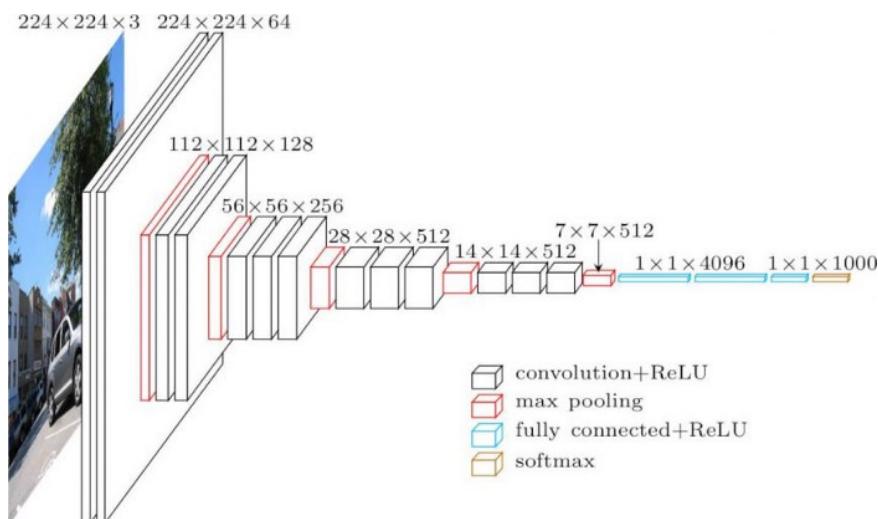


Figura 6.13.: Arquitectura de VGG-16. Imagen extraída de [Fei17].

Destacan las siguientes propiedades:

- Consigue en Imagenet una precisión del 92.7 %.
- Tiene unos 138 millones de parámetros entrenables que es más del doble de cualquiera de los modelos anteriores, lo que hace que VGG-16 sea muy lenta de entrenar.

#### 6.2.7.5. ResNet

Tras los modelos anteriores, todavía quedaba un problema que tenían las CNN sin resolver, y es que cuanto más profundas eran, más problema había con el cálculo de los gradientes, pues tendían a cantidades infinitesimalmente pequeñas. Es por ello que surge la arquitectura **ResNet** [HZRS16], que introduce cortes en la red conectando un tensor con el que había unas cuantas etapas atrás, a esto se le denomina **bloque residual**.

Con estos bloques residuales impedimos que los gradientes tiendan a cero rápidamente, con lo que podemos crear redes mucho más profundas. De hecho la versión de ResNet con 152 capas ganó el concurso ILSVRC 2015 siendo la red más profunda en aquel entonces.

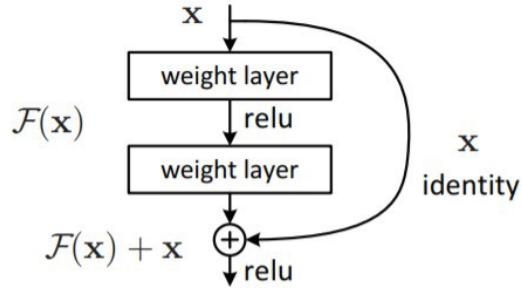


Figura 6.14.: Bloque residual de una ResNet. Como vemos, se suma el tensor  $x$  con el tensor  $\mathcal{F}(x)$  que surge unas etapas después. Imagen extraída de [HZRS16].

### ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

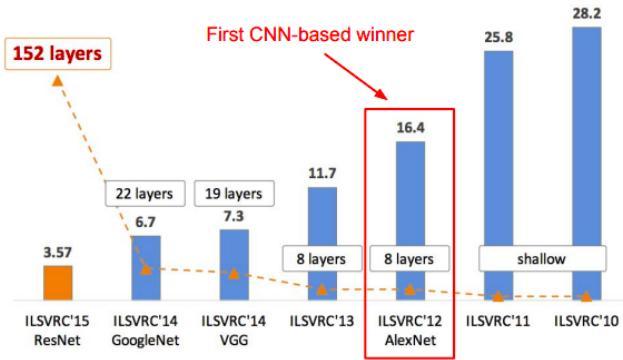


Figure copyright Kaiming He, 2016. Reproduced with permission.

Figura 6.15.: Resumen de los ganadores del concurso ILSVRC hasta 2015 con la aparición de Resnet. Imagen extraída de [Fei17].

## 6.3. Autoencoders

Hasta ahora hemos visto qué son las CNN y su gran importancia en Deep learning en tareas de reconocimiento de objetos en imágenes. A continuación vamos a presentar un tipo de red que se emplea en tareas de *Aprendizaje no supervisado* y que será de gran importancia en nuestro trabajo. Se tratan de las redes conocidas como **Autoencoders**.

### 6.3.1. Introducción

Los **Autoencoders** [Der17] son un tipo específico de red cuya entrada “coincide” con su salida. Se encargan de reducir las imágenes de entrada a un vector perteneciente a un espacio vectorial latente y que idealmente codifica los elementos más relevantes en la imagen para posteriormente reconstruir la imagen a partir de este vector con la intención de que sea lo más parecida posible a la de entrada.

Los autoencoders tienen tres componentes principales:

- **Encoder:** Suele ser una subred que se encarga de codificar la entrada a un vector de un espacio vectorial latente.
- **Code:** es el vector que codifica la imagen de entrada.
- **Decoder:** se trata de una subred que reconstruye la imagen a partir del vector.

Como podemos ver, no se necesita ningún tipo de etiqueta para reconstruir las imágenes de entrada, por lo que este tipo de redes se emplean en tareas de *Aprendizaje no supervisado*.

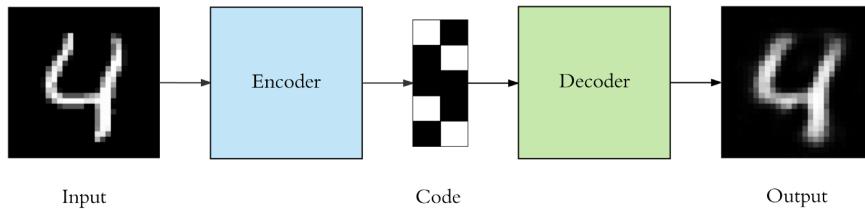


Figura 6.16.: Esquema de un Autoencoder básico. Imagen extraída de [Der17].

Esta estructura ha ido evolucionando durante los últimos años, por lo que vamos a ver a continuación esta evolución hasta llegar a la red que emplearemos en nuestro trabajo, el **Adversarial Autoencoder**.

### 6.3.2. Evolución de los Autoencoders

#### 6.3.2.1. Generative Adversarial Networks (GANs)

Es una red cuyo fin es “*crear nuevo contenido*” y que sea lo más similar posible al contenido de la base de datos que usamos para entrenar la red. Para ello se pretende aprender la distribución de probabilidad que siguen los píxeles de las imágenes del dataset, por lo que el objetivo realmente sería el re generar nuevos valores aleatorios de la distribución de probabilidad que siguen las imágenes [Roc19a].

Para esta tarea suele emplearse una CNN denominada *Generative Network*, que tiene como objetivo recibir una imagen o un tensor que sigue una distribución conocida y producir como salida una imagen de las mismas dimensiones que la de entrada que siga la distribución desconocida de las imágenes del dataset a nivel de **pixel**. Esta red tiene la estructura de un **Autoencoder**.

Por otro lado, se hace uso de una segunda CNN denominada *Discriminador*, que recibe un conjunto de imágenes del dataset e imágenes generadas por el *Genenerative Network* imagen de entrada y trata de clasificarlas en imágenes procedentes del dataset o imágenes generadas por la red. Por lo que el objetivo de la GAN en general podría resumirse en generar imágenes con la *Genenerative Network* que sean capaces de engañar al *Discriminador*, y para esto se pretende aprender lo mejor posible la distribución de probabilidad de las imágenes del dataset.

## 6. Fundamentos Teóricos y Métodos

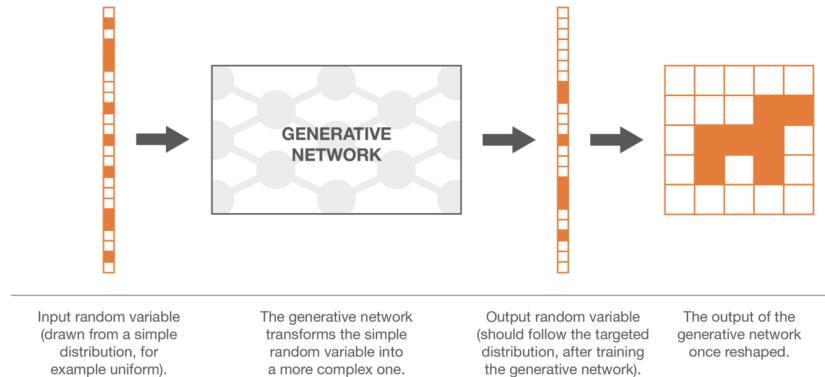


Figura 6.17.: Ejemplo de una Generative Network que pretende aprender la distribución de probabilidad de un conjunto de imágenes de perros. Imagen extraída de [Der17].

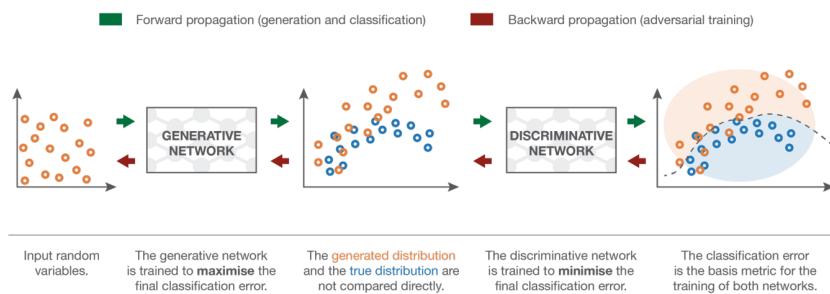


Figura 6.18.: Resumen del proceso de entrenamiento de una GAN. Imagen extraída de [Der17].

### 6.3.2.2. Variational Autoencoder (VAE)

Un VAE puede definirse como un autoencoder cuyo entrenamiento es regularizado para evitar el overfitting y asegurarse que el espacio vectorial latente tiene propiedades adecuadas para la generación de nuevos datos.

La principal diferencia entre el Autoencoder y el Variational Autoencoder es que en vez de codificar cada elemento de entrada como un punto del espacio vectorial latente, va a codificarlo como una distribución sobre el espacio latente buscando la distribución de los datos, no de los píxeles. El modelo de entrenamiento sería:

- La entrada se codifica como una distribución sobre el espacio vectorial latente.
- Un punto del espacio latente es muestreado por la distribución.
- Se decodifica el punto y se calcula el error de reconstrucción.
- Se usa backpropagation con el error anterior.

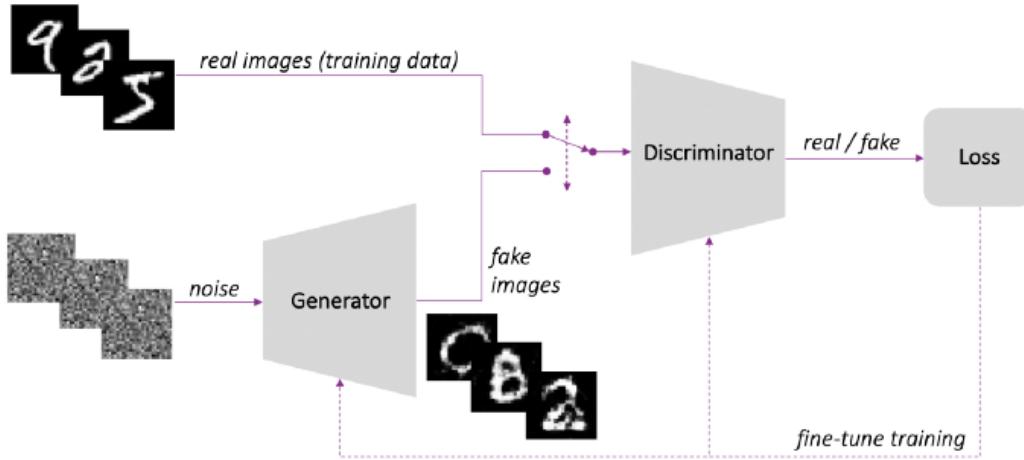


Figura 6.19.: Ejemplo de la arquitectura de una GAN para generar imágenes de dígitos manuscritos. Imagen extraída de [Ras20]

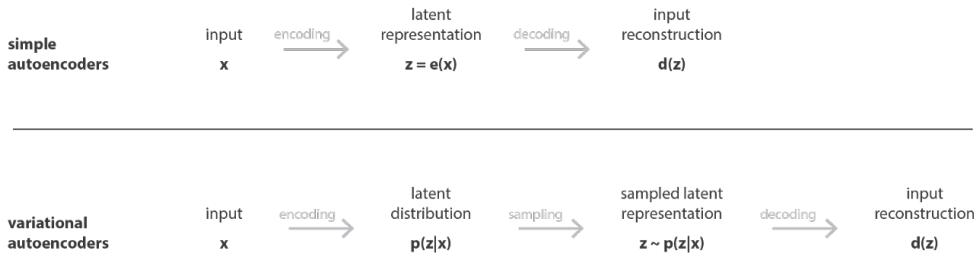
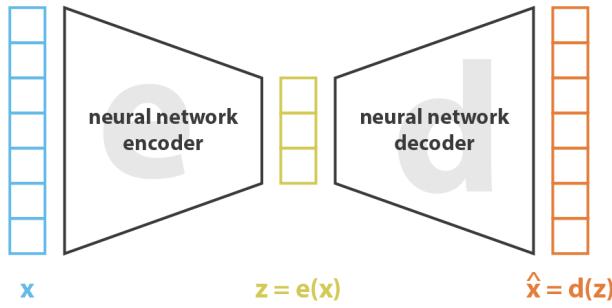


Figura 6.20.: Diferencia en el tratamiento de los datos por parte del encoder en una VAE y un Autoencoder clásico. Imagen extraída de [Roc19b].

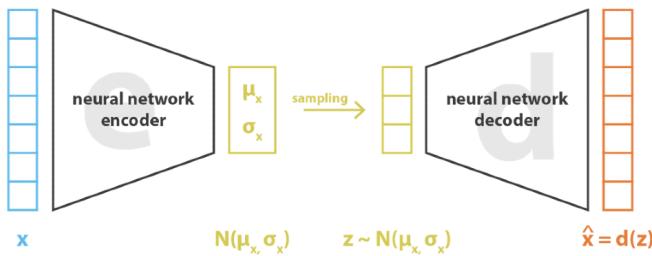
El modo de proceder habitual es tomar las distribuciones de los datos codificados como normales, así puede devolverse la media y matriz de covarianzas que describen la Gaussiana. La razón por la que la entrada se codifica como una distribución es porque esto permite expresar de forma natural la regularización del espacio vectorial latente ya que las distribuciones que salen del codificador se forzarán a que sean lo más similar posible a una distribución normal estandar, este término regularizante se incluye en la función de coste como podemos ver en Figura 6.21. Para ello utiliza la divergencia de *Kulback-Leibler* (KL) como medida entre la diferencia entre dos distribuciones.

### 6.3.2.3. Adversarial Autoencoder(AAE)

Un Adversarial Autoencoder no es más que la unión de la arquitectura de un Autoencoder con el concepto de Adversarial Loss y el Discriminante introducido por la GAN. Por otro lado utiliza la idea del VAE para regularizar el espacio vectorial latente pero en lugar de la divergencia de KL emplea el adversarial loss, y en lugar de muestrear una distribución



$$\text{loss} = \|x - \hat{x}\|^2 = \|x - d(z)\|^2 = \|x - d(e(x))\|^2$$



$$\text{loss} = \|x - \hat{x}\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)] = \|x - d(z)\|^2 + \text{KL}[N(\mu_x, \sigma_x), N(0, I)]$$

Figura 6.21.: Diferencia en la función de pérdida de un Autoencoder clásico (imagen superior) y una VAE (imagen inferior). Destacamos el término KL regularizante que pretende que las distribuciones de los datos sigan una normal estándar. Imagen extraída de [Roc19b].

con diferentes parámetros para cada imagen de entrada, lo que pretende es que todas las imágenes de entrada, al ser codificadas, sigan la misma distribución prefijada.

Para ello se añade un nuevo componente que actúa como Discriminador y el Encoder actuará como Generador(a diferencia de las GANs, dónde la salida del Generador era la imagen, no el vector del espacio vectorial latente). Se selecciona una distribución a seguir por los vectores del espacio vectorial latente (generalmente una distribución normal estándar). Así, los vectores generados por el Encoder tratarán de engañar al Discriminador, que tendrá que discernir entre si proceden de la distribución elegida o no. En otras palabras, si el vector latente es una muestra aleatoria de la distribución deseada o bien es un vector generado por el encoder.

Por lo tanto la arquitectura de un AAE presenta los siguientes componentes:

- **Encoder.** Tomará la entrada y la transformará en un vector latente de baja dimensión.
- **Decoder.** Tratará de reconstruir la imagen a partir del vector latente generado por el

Encoder.

- **Discriminador.** Toma vectores de la distribución del espacio vectorial latente deseada (reales) y también vectores generados por el Encoder (fakes) y tratará de discernir entre si proceden de la distribución o no.

Podemos ver un esquema de esta arquitectura en la [Figura 6.22](#). Esta arquitectura será la que emplearemos en nuestro proyecto, aunque presenta algunas variaciones que introduciremos más adelante.

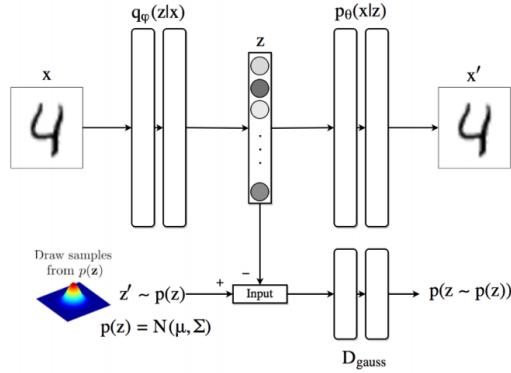


Figura 6.22.: Esquema de la arquitectura de un AAE. El Encoder sería la red a la que entra la imagen  $x$ , el vector  $z$  sería el vector latente, que sirve de entrada al Discriminador  $D_{\text{gauss}}$  y finalmente el vector  $z$  es la entrada del Decoder que reconstruye la imagen. Imagen extraída de [\[Ras20\]](#).

## 6.4. Técnicas empleadas

En esta sección vamos a presentar las distintas técnicas que se emplearán en el trabajo durante el entrenamiento de la red que utilizaremos y que se presentará más adelante.

### 6.4.1. Few-shot Learning y Data Augmentation

Entendemos por **Few-shot learning** los problemas de AA en los que en la etapa de aprendizaje supervisado se dispone de muy pocos datos etiquetados. En nuestro caso disponemos de un dataset Forense con pocas imágenes etiquetadas, por ello lo consideramos un problema de few-shot learning.

Por otro lado, cuando se presentan este tipo de problemas se aplican técnicas de **data augmentation**. Esta técnica consiste en *crear* nuevos datos de entrenamiento a partir de las imágenes originales. Para ello se aplican *deformaciones* a las imágenes del dataset, como *rotaciones*, *traslaciones* u *occlusiones*. Este tipo de transformaciones pueden ser de gran utilidad para entrenar el modelo en datos más complicados que los originales y para añadir variabilidad al dataset.



## 7. Estado del Arte

En esta sección nuestro objetivo será realizar una investigación por la literatura y los artículos publicados relacionados con el reconocimiento automático de landmarks cefalométricos en tareas de antropología forense para finalmente justificar nuestra elección en el presente trabajo para tratar de resolver el problema. Para ello utilizaremos la base de datos *Scopus* para realizar la búsqueda y consulta de artículos científico publicados.

### 7.1. Localización de landmarks cefalométricos en imágenes

Para hacernos una primera idea del estado actual del problema de reconocimiento de landmarks faciales en imágenes realizamos una primera consulta en *SCOPUS* con la siguiente *keyword* restringiendo los artículos a aquellos relacionados con la informática:

```
TITLE-ABS-KEY (facial AND (landmarks OR keypoints)AND detection ) AND  
(LIMIT-TO(SUBJAREA , "COMP"))
```

Tras esto realizamos una segunda búsqueda mucho más concreta al tipo de problema que tratamos de resolver. Usando la siguiente *keyword*:

```
TITLE-ABS-KEY (((anthropology OR (anthropology AND forensic)) AND  
(cephalometric AND (landmarks OR keypoints))) OR ((anthropology OR  
(anthropology AND forensic)) AND (facial AND (landmarks OR keypoints))))  
AND (LIMIT-TO(SUBJAREA,"COMP"))
```

Como podemos observar en las gráficas generadas para ambas *keywords* en la [Figura 7.1](#), actualmente existe una tendencia creciente en la publicación de papers relacionados con este tema, en particular esta tendencia comienza en los años en que surge el Deep Learning y las CNN comienzan a utilizarse en visión por computador para el tratamiento de imágenes. Sin embargo los artículos que se han encontrado no guardan una relación muy estrecha con el problema de detección de landmarks cefalométricos en problemas de antropología forense. Con la segunda búsqueda anterior se obtienen un total de 14 artículos, algo que nos confirma que es un área de investigación en la que apenas hay bibliografía o artículos.

#### 7.1.1. Evolución en la identificación forense de landmarks cefalométricos

Como hemos visto antes, la literatura existente es prácticamente nula, además de los catorce artículos obtenidos con la búsqueda anterior, la mayoría no se centran en el reconocimiento de landmarks cefalométricos en imágenes de rostros, hay artículos que se centran en la superposición craneofacial y otros basados en la identificación de landmarks en cráneos. Por lo tanto, de los catorce artículos vamos a destacar los siguientes por la relación directa con nuestro problema ordenados de mayor a menor antigüedad:

## 7. Estado del Arte

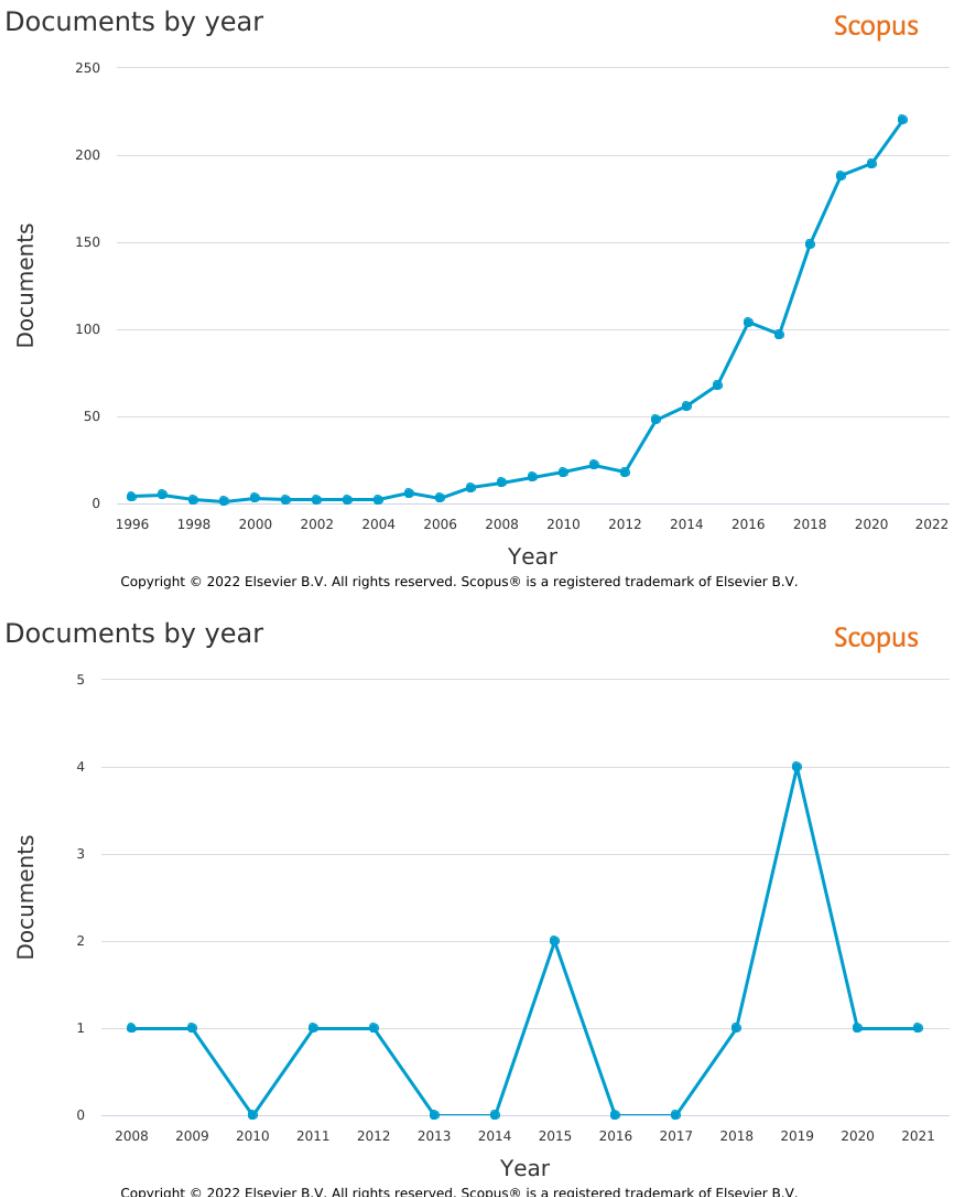


Figura 7.1.: Arriba podemos ver la gráfica de publicaciones por año obtenida con la primera keyword el 26 de Octubre de 2022, en total se encontraron 1,251 artículos. Destaca el notable incremento de papers a partir de 2012, año en que aparece la red AlexNet y comienza a ganar popularidad el Deep Learning en el tratamiento de imágenes. Abajo tenemos la gráfica de publicaciones por año obtenida con la segunda keyword el 26 de Octubre de 2022, en total se obtuvieron 13 artículos. Como vemos, existe una gran diferencia entre ambas a nivel de artículos publicados por año.

## 7.1. Localización de landmarks cefalométricos en imágenes

### 7.1.1. Automatic craniofacial anthropometry landmarks detection and measurements for the orbital region

Se trata de un artículo publicado en 2014 por Salina Mohd et al [AIA<sup>+</sup>14] en el que se pretende diseñar un método para calcular en una imagen de los ojos de un sujeto el *endocathion* y el *exocanthion* basado en el uso de un clasificador entrenado sobre filtros de Haar usados para la identificación de caras por Viola-Jones.

Lo primero que llama la atención del artículo es que tan solo pretende ser capaz de identificar dos landmarks, mientras que en nuestro problema por ejemplo tratamos de predecir la posición de unos treinta (incluyendo en *endocathion* y el *exocanthion*).

En segundo lugar, cabe destacar la manera en que se resuelve el problema, pues se hace uso de un clasificador en cascada basado en filtros de tipo Haar. Este es un método tradicional de la visión por computador en el que mediante el paso y convolución de este tipo de filtros por la imagen se obtiene información de esta relativa a los contornos. No obstante se trata de un método que ya se ha visto superado por otras técnicas más recientes de deep learning.

Por otro lado, el conjunto de datos que se emplea se ha obtenido en entornos controlados con buena iluminación, algo que no guarda relación con nuestro problema pues se trata de imágenes en diversas posturas, iluminación, y resolución.

### 7.1.2. Automated facial landmark detection, comparison and visualization

Se trata de un trabajo realizado en 2015 por Marek Galvánek et al. [GFCS15] para la detección automática de landmarks en modelos 3D de imágenes de personas.

Los landmarks detectados por el modelo son en total 14 y todos ellos pertenecen también al conjunto de landmarks que se detectan en este trabajo fin de grado.

El algoritmo propuesto se basa en la curvatura de la superficie del modelo 3D y en la simetría del perfil. En primer lugar se alinea el modelo 3D con el plano horizontal de Frankfort Figura 7.2, un plano utilizado por los antropólogos forenses para marcar landmarks. En segundo lugar se realiza un estudio de la curvatura del modelo 3D en la zona de la nariz, boca y ojos. Finalmente con el perfil del modelo y la simetría se rectifican y perfeccionan los landmarks marcados en etapas previas.

El algoritmo propuesto resulta interesante, aunque no detecta un elevado número de landmarks y se hace de una forma parecida a como un antropólogo forense actuaría. Podemos ver también que es muy preciso en la detección si comparamos los landmarks marcados por el algoritmo con los marcados por un experto manualmente en la Figura 7.3.

En este trabajo, comenzamos a ver ya un problema similar al nuestro, la detección automática de landmarks cefalométricos con un mayor número de landmarks (14 frente a los 2 del estudio anterior). Aunque se realiza sobre modelos 3D de caras que evitan problemas de oclusión, mala iluminación o resolución.

### 7.1.3. Automatic cephalometric landmarks detection on frontal faces: An approach based on supervised learning techniques

El siguiente artículo es de 2019 y fue realizado por Lucas Faria et al. Pretende desarrollar un algoritmo para el reconocimiento automático de landmarks cefalométricos en imágenes fron-

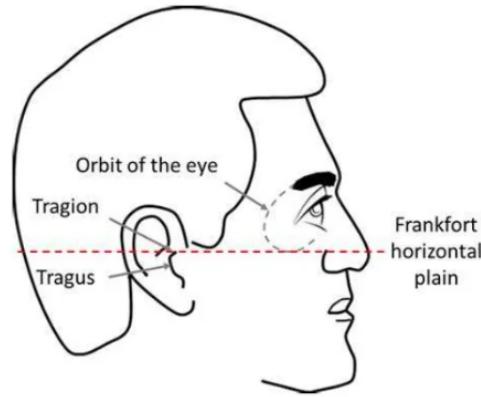


Figura 7.2.: Cara alineada con el plano horizontal de Frankfort. Imagen extraida de <https://www.slideshare.net/NiharikaSupriya/cephalometrics-landmarks-lines-and-planes-93890774>

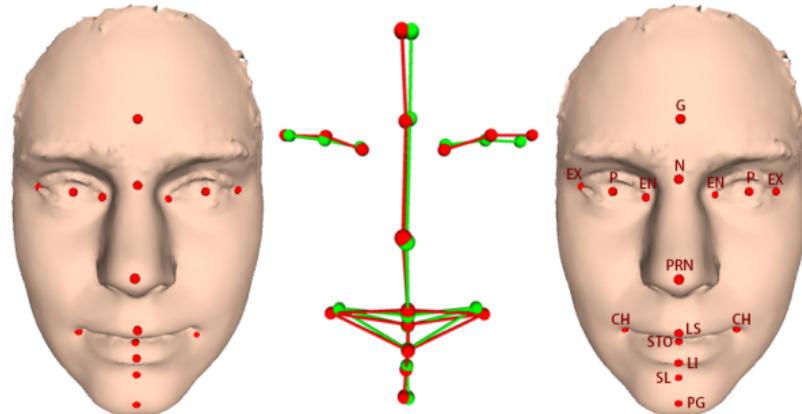


Figura 7.3.: Comparativa entre los landmarks marcados por el algoritmo desarrollado en el artículo (izquierda) y los marcados por un experto (derecha). Imagen extraida de [GFC15].

tales a partir de técnicas de visión por computador y de métodos de aprendizaje supervisado [PLF<sup>+</sup>19].

El algoritmo tiene tres componentes:

- En la primera fase se realiza un pre-procesamiento de las imágenes para resaltar sus características faciales.

## 7.1. Localización de landmarks cefalométricos en imágenes

- En la segunda fase se aplica la cascada de filtros de Haar de Viola-Jones para identificar las regiones de interés de la imagen: ojos, nariz, boca.
- Finalmente se aplica el algoritmo de machine learning supervisado a cada una de las regiones anteriores. Creando un detector automático de landmarks para cada región.

Por otro lado el conjunto de entrenamiento es de 1000 individuos de los cuales se tomaron fotografías frontales en las mismas condiciones de iluminación y de distancia a la cámara, por lo que no presenta las mismas complicaciones que el dataset del que disponemos.

Destaca este artículo por ser el primero en el cual comienzan a usarse técnicas de aprendizaje automático para la resolución del problema. Además en las imágenes de entrenamiento fueron marcados 28 landmarks que se emplearon para el entrenamiento, que coinciden con los mismos que tenemos en nuestro problema.

### 7.1.4. The Improved Faster R-CNN for Detecting Small Facial Landmarks on Vietnamese Human Face Based on Clinical Diagnosis

Este artículo es el más reciente pues fue publicado en Junio de 2022 por Ho Nguyen Anh Tuan et al [HNATT22]. En él se utiliza una versión mejorada de la red faster R-CNN aplicada a la tarea del reconocimiento de landmarks cefalométricos.

Los resultados obtenidos son muy buenos pero como ocurre en la mayoría de trabajos de este tipo, la base de datos usada ha sido de imágenes tomadas de voluntarios en unas mismas condiciones de iluminación frontales y de perfil.

No obstante el trabajo destaca la gran importancia que está teniendo el Deep Learning y las CNN en tareas de reconocimiento de landmarks faciales (usualmente landmarks no biológicos como los que se emplean en tareas de antropología forense), y partiendo de esta base podemos justificar el trabajo que vamos a desarrollar, pues nos proponemos adaptar una red que ya ha sido entrenada para la identificación de landmarks faciales en grandes volúmenes de datos de imágenes en diversas posturas y resolución para la tarea del reconocimiento de landmarks forense.

Todos los métodos que se han presentado en esta sección trataban de cumplir con el mismo objetivo, y han ido evolucionando con el paso de los años a la par que la informática, empezando por tratar de aplicar algoritmos coevolutivos, después filtros de Haar y técnicas de visión por computador y finalmente usar CNN de Deep Learning. De esta manera y viendo con perspectiva el estado del arte en el campo, consideramos que la propuesta que presentamos puede traer buenos resultados, pues pretendemos enseñar a un sistema experto en el reconocimiento de landmarks no biológicos a identificar estos otros puntos, lo cual puede suponer un nexo de unión entre las dos líneas de investigación.



## 8. Implementación

### 8.1. Diseño del Sofware

El diseño software no se ha orientado a crear un programa funcional para ser empleado por un usuario, sino que tiene como función entrenar los modelos probados y recabar los resultados oportunos del proceso de experimentación. Se ha utilizado el sistema de control de versiones *Git* junto con *GitHub*. En total se han creado dos repositorios:

1. Un repositorio con la planificación y elaboración de la memoria junto con los scripts utilizados para el preprocesamiento de los datos: <https://github.com/alejbormeg/TF-G-LandmarkDetectionAndCNNAnalysis>
2. Un *fork* a partir del repositorio principal del paper, donde se han elaborado todas las adaptaciones a nuestro problema: <https://github.com/alejbormeg/3FabRec/tree/master>

En primer lugar, se ha programado un *notebook* de Jupyter independiente al proyecto general del framework, en el cual, se realiza el preprocesamiento de los datos, la identificación de las caras y la separación en conjunto de entrenamiento y test. El fichero se denomina *Make-crop-split database.ipynb*.

Por otro lado, en lo que respecta al entrenamiento y diseño de la red, partíamos de un proyecto ya elaborado minuciosamente que se ha tenido que adaptar para aceptar como entrada la nueva base de datos así como alterar ciertas partes de su entrenamiento (para poder hacer el ajuste fino del decoder por ejemplo, realizar técnicas de *cross validation 5-fold*) o introducir nuevos parámetros y funciones para la obtención y manipulación de los datos, y la generación de las gráficas y tablas de resultados que se muestran. Además se tuvieron que alterar las funciones del cálculo de métricas para que tuvieran en cuenta que en todas las imágenes no necesariamente se incluían todos los landmarks, y por lo tanto que solo computasen aquellos presentes en las mismas.

La estructura de paquetes la podemos ver en la [Figura 8.1](#). Los paquetes en color rojo son los que han sido modificados o creados para este trabajo, el resto pertenecen al software original de [BW20]:

- **3FabRec.eval\_landmarks:** Se lleva a cabo la evaluación del modelo en el conjunto de test y se extraen las métricas. Se adapta para extraer el RMSE por landmark teniendo en cuenta que cada landmark puede estar no marcado en algunas imágenes.
- **3FabRec.train\_aae\_landmarks:** Se lleva a cabo todo el proceso del entrenamiento supervisado del marcado de landmarks. Es el fichero que más se ha tenido que modificar, pues hemos cambiado la lógica del entrenamiento con respecto a la del framework original. Los parámetros de ejecución se encuentran en la [Tabla 8.1](#).

## 8. Implementación

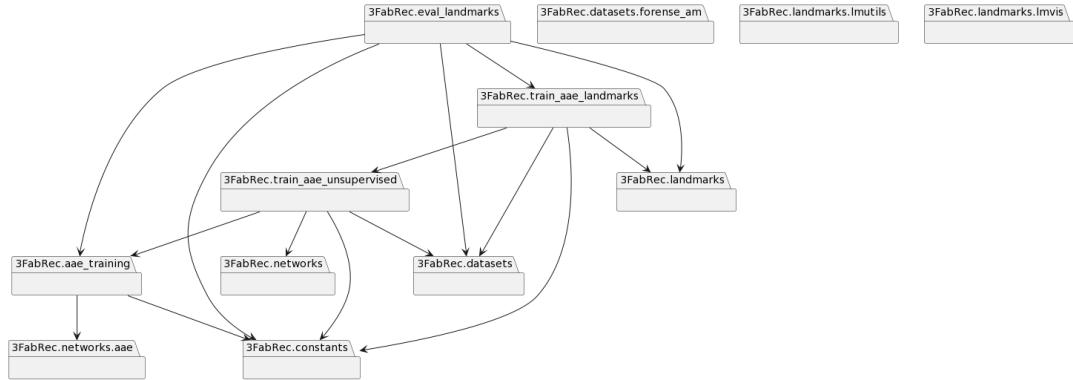


Figura 8.1.: Diagrama de paquetes del proyecto generado por *pyreverse*, herramienta incluida en el paquete Pylint

- **3FabRec.train\_aae\_unsupervised:** En este paquete se lleva a cabo el entrenamiento no supervisado de la red.
- Los dos paquetes anteriores, a su vez se combinan en **3FabRec.ae\_training**, que encapsula toda la lógica del entrenamiento, incluyendo el aprendizaje supervisado y el no supervisado.
- El paquete **3FabRec.landmarks** lleva a cabo todo lo relativo a la manipulación de los landmarks, el cálculo de los errores asociados a los mismos, el paso de Heat maps a coordenadas y la inversa, etc... En este fichero se ha tenido que modificar el subpaquete **3FabRec.landmarks.lmutils** y **3FabRec.landmarks.lmvis**. En el caso del primero porque hemos modificado algunas funciones que calculan las métricas de error adaptándolas al problema y en el caso del segundo porque no toda la información que se mostraban en las imágenes de salida era relevante para nuestro estudio, por lo que se han eliminado los datos innecesarios.
- El paquete **3FabRec.constants** contiene variables globales relativas a los conjuntos de datos de entrenamiento, validación y test.
- El paquete **3FabRec.networks** contienen todas las redes que se emplean en la arquitectura de la red. En concreto usaremos **3FabRec.networks.aae** en la cual se encuentra la estructura general del adversarial autoencoder.
- El paquete **3FabRec.datasets** contiene todos los ficheros relativos a la creación de la clase *Dataset* y *Dataloader* para cada base de datos concreta que se emplea en el framework. Dentro de este paquete se crea el fichero **3FabRec.datasets.foreNSE\_am**, que se encarga de la lectura correcta de los datos de entrenamiento de la base de datos proporcionada.

Por otro lado, en la Figura 8.3 podemos ver el diagrama de secuencia de la ejecución del software 3FabRec para la etapa de entrenamiento y validación y posteriormente para la de evaluación del modelo final. No obstante, remarcamos el hecho de que el software diseñado no tiene como fin su uso por parte de terceros. De ser así, se habría creado una interfaz de usuario amigable en la cual, de manera intuitiva se pudieran elegir las distintas opciones de ejecución.

## 8.2. Entorno de ejecución

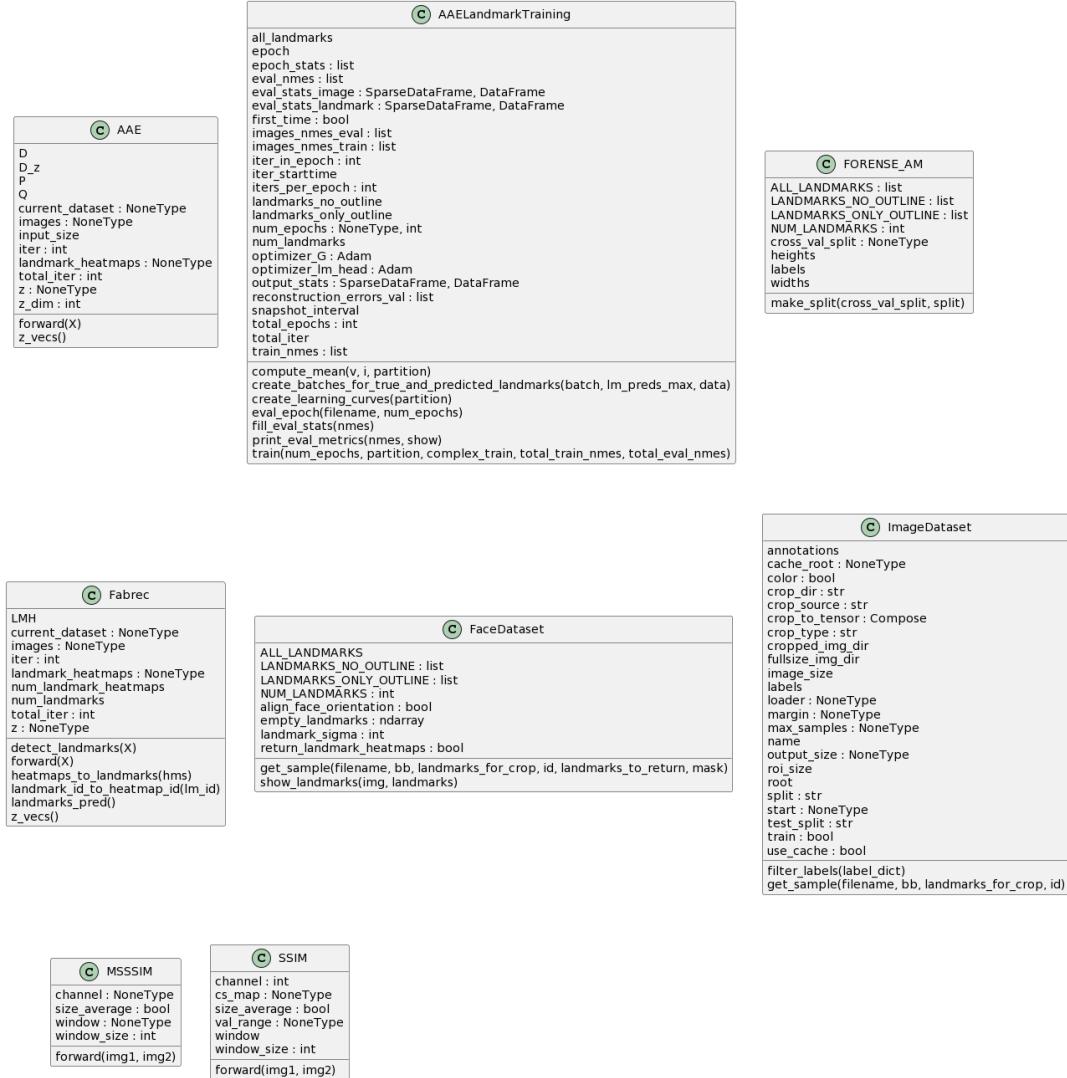


Figura 8.2.: Diagrama de clases con las principales clases del proyecto.

## 8.2. Entorno de ejecución

Las ejecuciones se han realizado todas en un ordenador portátil *HP Pavilion* con las siguientes características:

- Ubuntu 20.04.5 LTS 64 bits.
- 8 GB de RAM
- 8 x Intel® Core™ i5-8300H CPU @ 2.30GHz
- GPU NVIDIA GeForce GTX 1050 Mobile

Por otro lado, las versiones del software empleado son:

## 8. Implementación

Tabla 8.1.: Argumentos con los que se ha experimentado en la ejecución del fichero train-aae-landmarks.py

Parámetro	Descripción	Valor por defecto
-train-encoder	Si es True reentrena el encoder	False
-train-dencoder	Si True reentrena el dencoder	False
-epochs	Número de épocas que queremos entrenar la red	None
-batchsize	Tamaño del batchsize para entrenamiento	50
-batchsize-eval	Valor del batchsize para validación	10
-lr	Learning rate para el autoencoder	0.00002
-lr-heatmaps	Learning rate para las ITLs	0.001
-beta1	Valor de beta 1 para Adam	0.0
-beta2	Valor de beta 2 para Adam	0.999
-save-freq	Frecuencia de snapshot (en épocas)	1
-dataset	Dataset que se empleará	W300
-sigma	Tamaño de los heatmaps	7

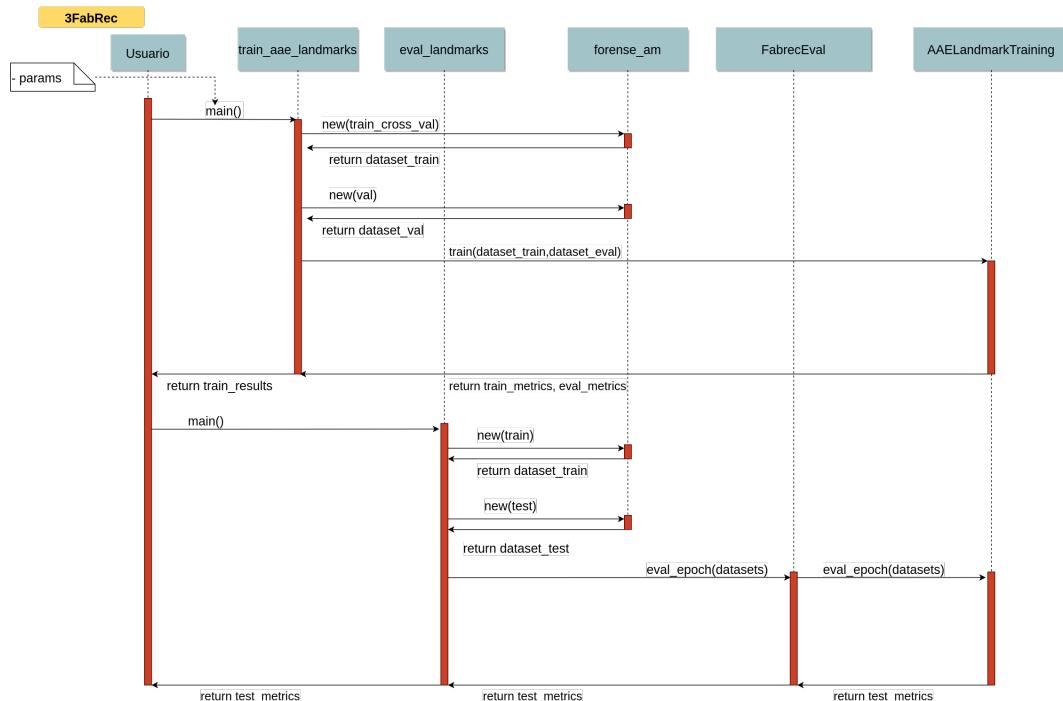


Figura 8.3.: Diagrama de secuencia del software empleado.

- Python 3.6.13
- CUDA 10.1
- PyTorch 1.1.0
- Numpy 1.17.4

## *8.2. Entorno de ejecución*

- Pandas 0.23.3
- Matplotlib 3.3.4



## 9. Solución propuesta y experimentos realizados

El reconocimiento automático de landmarks faciales es una área de gran importancia en la actualidad en tareas como el reconocimiento de personas y que se está viendo muy desarrollada gracias a las CNN. La principal diferencia entre este enfoque y el forense radica en el tipo de landmarks que se utilizan. Los utilizados en antropología forense son landmarks con justificación biológica. Dichos landmarks se corresponden con localizaciones concretas del cráneo del ser humano, y los landmarks cefalométricos intentan predecir la posición de estos puntos sobre la piel, a diferencia de los que se emplean en las tareas de reconocimiento facial, que generalmente atienden a puntos de interés de la cara para su correcto reconocimiento y son independientes del cráneo del sujeto. Así pues existen diversas bases de datos empleadas para este cometido con multitud de imágenes etiquetadas, destacamos entre ellas:

- **300-W**: Se trata de un dataset compuesto por 3148 imágenes de entrenamiento y 689 imágenes de test *in-the-wild*, es decir con multitud de poses, distinta iluminación y expresiones faciales. Está anotada por 51 landmarks o 68 landmarks si contamos el contorno del rostro [STZP13]. Podemos ver los landmarks en la imagen Figura 9.1
- **AFLW**: Se trata de un dataset de 24386 imágenes *in-the-wild* con un total de 21 landmarks anotados entre las cejas y el mentón, como podemos ver en la imagen Figura 9.2 [KWRB11]

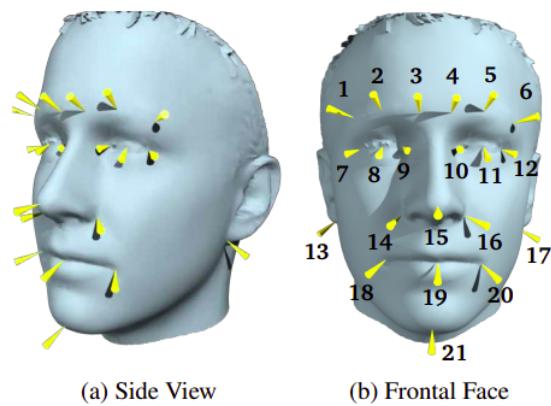


Figura 9.2.: Conjunto de landmarks anotados sobre un modelo 3D que emplea el dataset AFLW. Imagen extraída de [KWRB11].

El hecho es que existen actualmente CNN que son capaces de reconocer con un alto grado

## 9. Solución propuesta y experimentos realizados

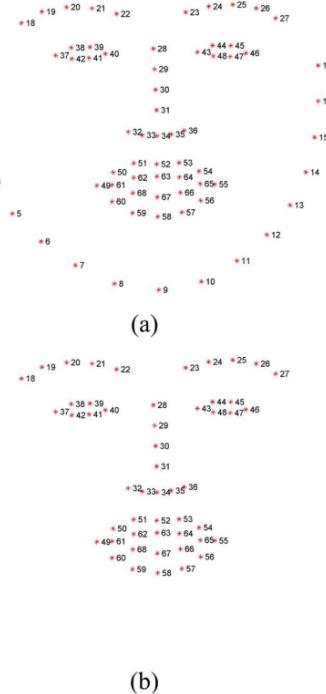


Figura 9.1.: Conjunto de landmarks anotados en el dataset 300W, en la imagen *a* contando el contorno del rostro son un total de 68 landmarks, en la imagen *b* son 51 en total. Imagen extraida de [STZP13].

de precisión estos conjuntos de landmarks marcados por las bases de datos anteriores, lo que nos hace pensar que quizá podría emplearse el conocimiento adquirido por estas redes para reentrenarlas en un proceso de *fine-tuning* sobre una base de datos forense con landmarks anotados por un experto para tratar de resolver el problema de la identificación automática de landmarks. De ahí nace nuestra propuesta, que en cierto modo trata hace como nexo de unión entre las dos vías de investigación.

### 9.1. Framework empleado: 3FabRec

La red empleada para la resolución del problema es la desarrollada por Bjorn Browatzki et al en 2020 [BW20] denominada **3FabRec**. La red en cuestión es un **Adversarial Autoencoder** combiando con una red *GAN* (por la presencia de un segundo discriminante propio de las *GANs*) que puede predecir landmarks gracias a la incorporación de unas capas convolucionales intermedias denominadas *Interleaved Transfer Layers* (*ITLs*) en la etapa de reconstrucción. Todos estos elementos serán explicados en profundidad más adelante.

Esta red aplica un método *semi-supervisado* en el cual:

- Hay una primera fase de **aprendizaje no supervisado**, donde se pretende adquirir conocimiento implícito sobre la estructura facial contenida en grandes conjuntos de

imágenes de rostros de personas en diversas posiciones, iluminación y etnia. Para ello se codifica todo este conocimiento implícito en un vector de un espacio latente de baja dimensionalidad para posteriormente reconstruir la imagen. Este proceso se hace íntegramente en el *Adversarial Autoencoder*, sin hacer uso de las ITLs.

- Posteriormente, en una segunda fase de **aprendizaje supervisado**, se entrena la red con un conjunto de imágenes etiquetadas con landmarks faciales que la red tratará de predecir. Para ello se intercalan entre las capas del generador capas de convolución. Estas son las ITLs que mencionamos anteriormente, y se encargan de reutilizar el conocimiento estructural aprendido y fijado en la primera fase del entrenamiento para adaptar el generador a la tarea de predecir los mapas de calor.
- Finalmente, se puede incluir una tercera fase de *finetuning* en la cual se entrena el Encoder para mejorar el rendimiento en la predicción de landmarks.

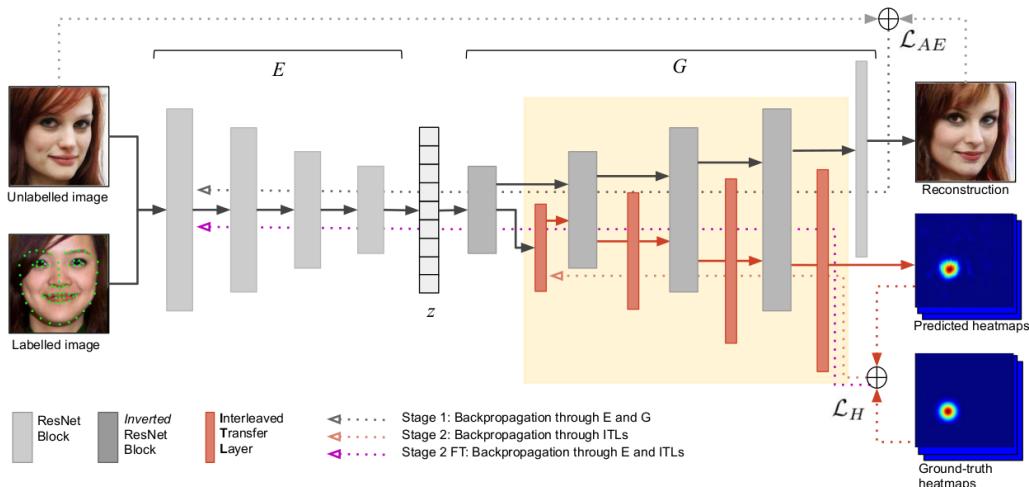


Figura 9.3.: Imagen resumen del framework 3FabRec. En ella podemos ver la estructura del *Adversarial Autoencoder*, dividido en un Encoder (región bajo la E) y un Generator (región bajo la G). En rojo podemos ver las ITLs que se intercalan entre cada capa del Generador y dan como resultado un conjunto de mapas de calor. Imagen extraída de [BW20].

### 9.1.1. Arquitectura Adversarial Autoencoder

Para la construcción del *Adversarial Autoencoder* utilizan:

- **Encoder:** emplean una ResNet-18 hasta codificar la entrada en un vector de 99 dimensiones. Está pensado para imágenes de res  $256 \times 256 \times 3$ , aunque se adapta también a imágenes de dimensiones  $512 \times 512 \times 3$ .
- **Decoder:** emplean la misma red ResNet-18 pero invertida.

## 9. Solución propuesta y experimentos realizados

Para una mejor comprensión hemos realizado unos diagramas con la herramienta *diagrams.net*. En la [Figura 9.4](#) podemos ver la estructura básica de los bloques de la ResNet-18. Por otro lado en la [Figura 9.5](#) podemos ver el paso de una imagen de entrada de tres canales y resolución  $256 \times 256$  por el encoder.

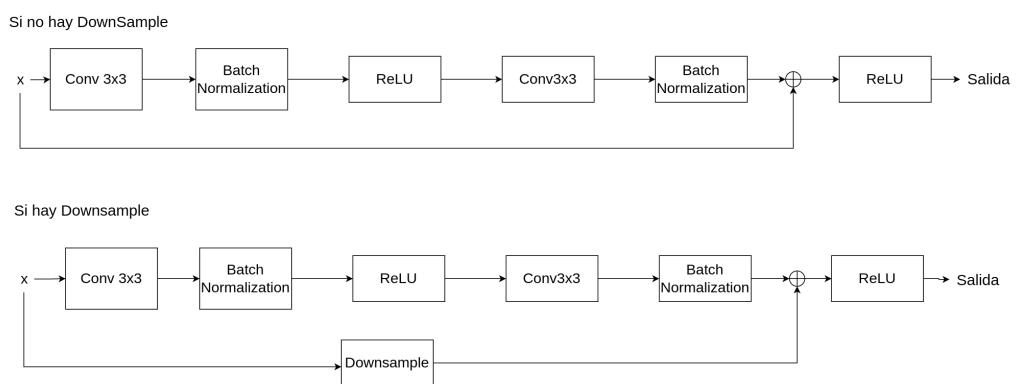


Figura 9.4.: Bloques básicos que utiliza la red ResNet-18 en sus capas. Se trata de una sucesión clásica de Convolución 3x3 + Batch Normalization + ReLU que se repite dos veces. En el primer caso los filtros de convolución no reducen las dimensiones del tensor añadiendo un padding de 1. En el segundo caso se reduce la dimensión del tensor a la mitad tras la primera convolución y se mantiene la dimensionalidad en la segunda. En el primer caso, la suma residual puede realizarse con el tensor  $x$  sin problema, en el segundo caso el tensor debe reducirse para que casen las dimensiones.

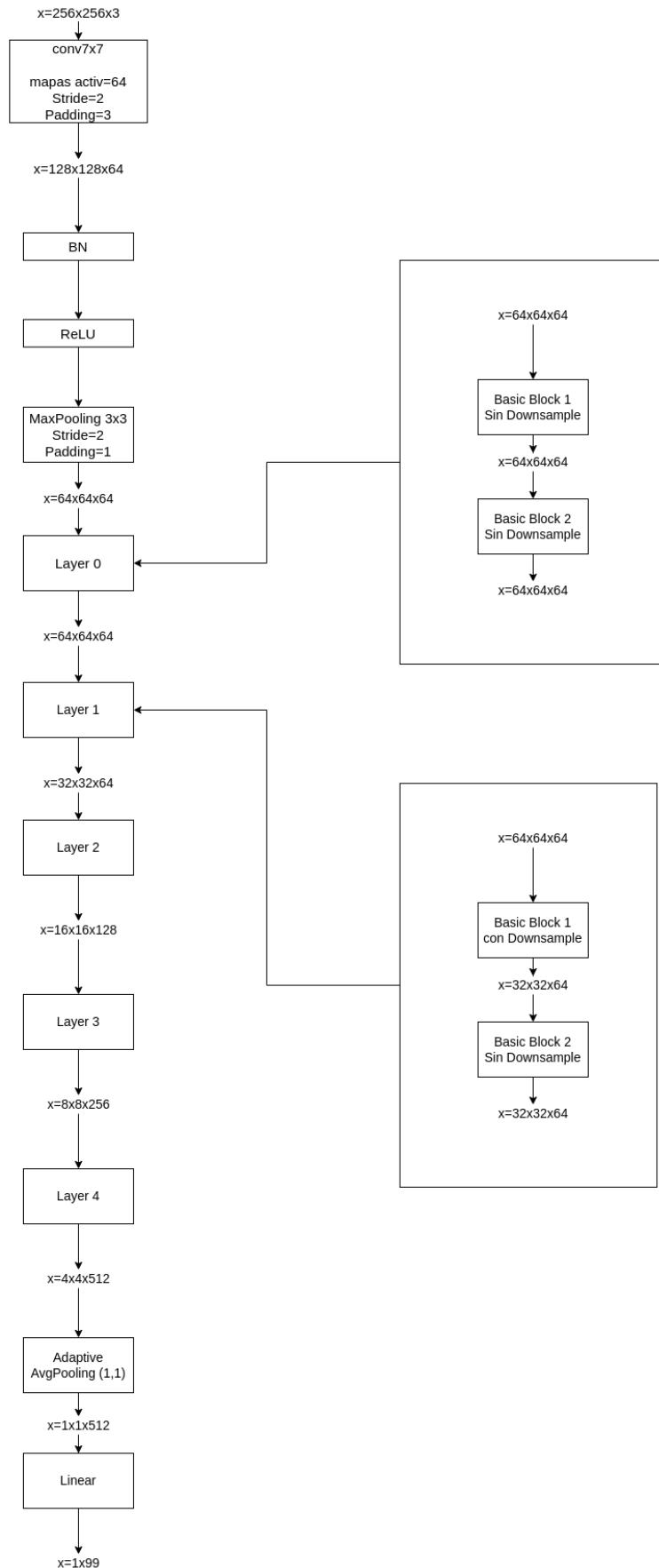
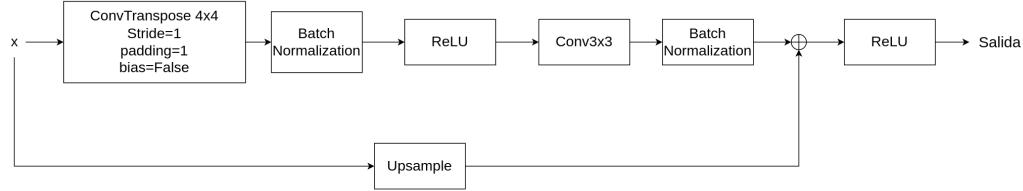


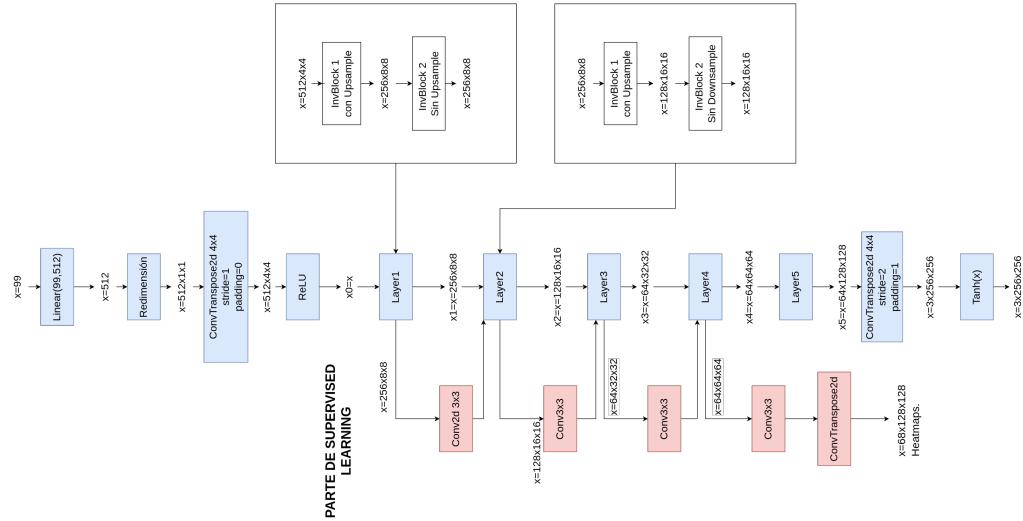
Figura 9.5.: Ejemplo de paso de una imagen a través del Encoder. Cabe destacar que a partir de la Layer 1, todos los bloques tienen downsample.

## 9. Solución propuesta y experimentos realizados

En la [Figura 9.6](#) podemos ver la estructura básica de un bloque en el Generador *Inverse ResNet*, y un ejemplo del paso de un vector por el generador podemos verlo en la [Figura 9.7](#)



[Figura 9.6](#).: En primer lugar se aplica una convolución transpuesta que duplica las dimensiones del tensor de entrada y tras esto se sigue la misma estructura que en el bloque básico de la ResNet-18, la segunda convolución  $3 \times 3$  mantiene las dimensiones. Como consecuencia, para sumar el tensor de entrada con la salida del bloque se aumentan las dimensiones de este.

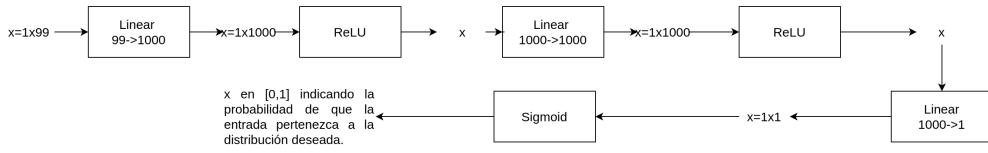


[Figura 9.7](#).: Ejemplo del paso de un vector de 99 dimensiones por el generador hasta reconstruirse la imagen de dimensiones  $256 \times 256 \times 3$ . La parte correspondiente al aprendizaje supervisado es la de los cuadrados azules, los cuadrados rojos corresponden a las ITLS de la parte supervisada que se intercalan entre cada dos capas y dan como resultado los mapas de calor de los landmarks predichos.

Finalmente, necesitamos un Discriminador, que procure que los vectores del espacio vectorial latente sigan una determinada distribución. Dicha distribución será una normal multivariante estándar. Por otro lado, se añadirá un segundo discriminador propio de las redes GAN que nos dirá si la imagen reconstruida procede de la distribución que siguen los píxeles de la imagen de inicio. Este discriminador permite que la red genere imágenes más realistas. En la [Figura 9.8](#) las redes neuronales que definen ambos discriminadores.

## DNet\_Gaussian

Durante el entrenamiento se usa un Dropout de 0.2



## Discriminante

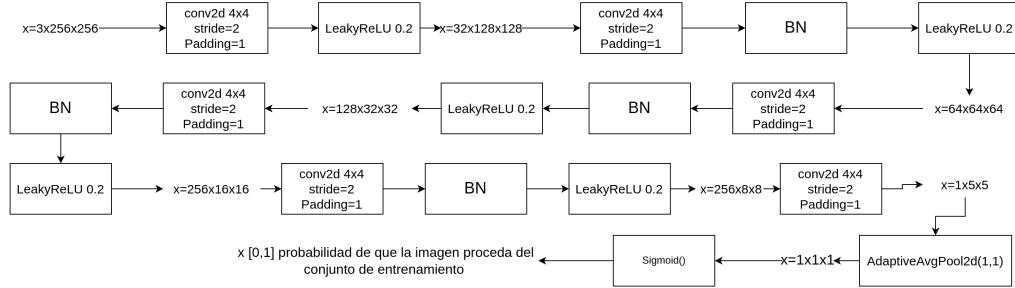


Figura 9.8.: En la imagen superior vemos el discriminante que se emplea para los vectores producidos por el Encoder y en la imagen inferior vemos el discriminante que se emplea para las imágenes generadas por el Generador. En ambos casos se da como salida un valor entre 0 y 1 que hace referencia a la probabilidad de pertenecer a la distribución deseada en el primer caso o a seguir la distribución de los píxeles de las imágenes en el segundo caso.

### 9.1.1.1. Interleaved Transfer Layer (ITL)

Se denominan ITLs a las capas convolucionales encargadas de la predicción de landmarks. Se tratan de una solución novedosa porque premiten adaptar una AAE pensado originalmente para tareas de aprendizaje no supervisado a un problema de **regresión**, típico del aprendizaje supervisado.

Se tratan de simples capas convolucionales que se intercalan entre las capas del Generador. La última de estas capas proporciona como salida un conjunto de mapas de calor, uno por cada landmark predicho. Estos mapas de calor luego se emplean para representar en la imagen reconstruida los landmarks. La arquitectura de esta etapa podemos verla en la Figura 9.7.

### 9.1.2. Función de pérdida

La red utiliza dos funciones de pérdida, la primera que presentaremos se emplea en el entrenamiento del *Adversarial Autoencoder*, en la parte no supervisada, y la segunda se utiliza tanto en la parte de aprendizaje supervisado como en la de *fine-tuning* del Encoder. Debido al interés que tiene en el estudio del framework se van a presentar y explicar ambas funciones de pérdida. Cabe resaltar, que debido a que el problema que vamos a resolver es de **regresión**, en la fase de experimentación únicamente tendrá interés la segunda función de coste, asociada al entrenamiento de la parte supervisada.

## 9. Solución propuesta y experimentos realizados

La función de pérdida empleada para el entrenamiento del *Adversarial Autoencoder* en la parte de aprendizaje no supervisado es la siguiente:

$$\begin{aligned} \min_{E,G} \max_{D_z, D_x} \mathcal{L}_{AE}(E, G, D_z, D_x) = \\ \lambda_{rec} \mathcal{L}_{rec}(E, G) + \lambda_{cs} \mathcal{L}_{cs}(E, G) \\ + \lambda_{enc} \mathcal{L}_{enc}(E, D_z) + \lambda_{adv} \mathcal{L}_{adv}(E, G, D_x) \end{aligned}$$

En la cual  $\lambda_{enc}$  y  $\lambda_{adv}$  toman el valor 1.0 mientras que  $\lambda_{rec}$  y  $\lambda_{cs}$  se establecen en 1.0 y 60.0 respectivamente. El valor de la función de coste se propagará por los pesos de  $E, G, D_z$  y  $D_x$  actualizándolos mediante *back-propagation*.

Como podemos observar en la expresión anterior, se trata de una combinación lineal de cuatro funciones de coste distintas:

- *Error de reconstrucción*: Se trata de un error encargado de asegurarse de que la red reconstruya de forma apropiada las imágenes, para ello calcula la distancia  $L_1$  entre la imagen original y la reconstruida pixel a pixel. Tiene la siguiente expresión, que coincide con la presentada en el apartado anterior de métricas.

$$\mathcal{L}_{rec}(E, G) = \mathbb{E}_{x \sim p(x)} [\|x - G(E(x))\|_1]$$

- *Error estructural de la imagen*: Se trata de una variante del **SSIM** encargado de que la imagen real y la reconstruida tengan una estructura similar. La expresión de la función  $cs$  que aparece en la fórmula se detalla más adelante.

$$\mathcal{L}_{cs}(E, G) = \mathbb{E}_{x \sim p(x)} [cs(x, G(E(x)))]$$

- *Error del Encoder*: es el error empleado comúnmente en este tipo de redes y que se asegura que el vector latente que genera el encoder siga la distribución impuesta en el discriminador  $D_z$ . Su expresión es:

$$\mathcal{L}_{enc}(E, D_z) = \mathbb{E}_{z^* \sim p(z)} [\log(D_z(z^*))] + \mathbb{E}_{x \sim p(x)} [\log(1 - D_z(E(x)))]$$

- *Error adversario de imagen*: normalmente las imágenes generadas por los Autoencoders suelen generar imágenes borronadas. Es por ello que se introduce este error, que añade un discriminador típico de una red *GAN* que pretende discernir entre si la imagen que recibe como salida de la red pertenece al conjunto de imágenes de entrenamiento o si es una imagen generada por la red, de esta forma se generan imágenes más realistas. Su expresión es la siguiente:

$$\mathcal{L}_{adv}(E, D_x) = \mathbb{E}_{x \sim p(x)} [\log(D_x(x))] + \mathbb{E}_{x \sim p(x)} [\log(1 - D_x(G(E(x)))]$$

Por otro lado, la función de pérdida que se empleará en el **entrenamiento de las ITLs** será la siguiente:

$$\mathcal{L}_H(ITL) = \mathbb{E}_{x \sim p(x)} [\|H - ITL(a)\|_2]$$

Dónde  $a$  serían los mapas de activación que genera la ResNet invertida para la imagen codificada  $z = E(x)$  (siendo  $x$  la imagen de entrada a la red). En este caso se computa la distancia  $L_2$  entre los *Heat-maps* de los landmarks originales de la imagen  $x$  de entrada, y los predichos por las *ITLs*. Propagando el error por los pesos de las *ITLs* solamente, forzando a adaptarse a los pesos ya aprendidos del generador.

A modo de aclaración, las imágenes etiquetadas con landmarks a la red suelen proporcionarse con el siguiente formato:

- Un archivo con la imagen sin etiquetar.
- Un archivo de texto plano con las coordenadas de cada landmark en la imagen.

Dados estos archivos, el framework, calcula para cada landmark proporcionado un mapa de calor en una imagen de tamaño  $128 \times 128$  en el caso de que las imágenes de entrada sean de  $256 \times 256$ . Y es a ese conjunto de imágenes (una por cada landmark) a las que denominamos  $H$  en la función anterior.

Finalmente, en la etapa de *fine-tuning* se computa la misma función de pérdida de antes, con la salvedad de que el error se propaga tanto por las *ITLs* como por los pesos del *Encoder*. Esto permite que el *Encoder* se codifiquen con mayor precisión las imágenes y que se eliminan factores irrelevantes para la predicción de landmarks como son el género, el color de piel o la iluminación. Por otro lado se evita el overfitting ya que durante esta última etapa los pesos del *Decoder* no se actualizan.

### 9.1.3. Proceso de entrenamiento de la red

#### 9.1.3.1. Entrenamiento no-supervisado

El framework que empleamos ha sido entrenado durante 50 épocas con imágenes de tamaño  $256x256$  y con un tamaño de batch de 50. En todas las etapas del entrenamiento se ha empleado un optimizador tipo Adam, el cual durante el entrenamiento del *Adversarial Autoencoder* usó  $\beta_1 = 0.0$  y  $\beta_2 = 0.999$  con un *learning rate* de  $2 \times 10^{-5}$ .

Por otra parte, se aplicaron técnicas de *data-augmentation* a las imágenes de entrada como giros horizontales, traslaciones, *resizing* o rotaciones.

#### 9.1.3.2. Entrenamiento supervisado

Para el entrenamiento de la parte supervisada, las imágenes de entrada se recortan de acuerdo a un *bounding-box* creado por el framework a partir de unas coordenadas de entrada o bien de acuerdo a los landmarks que se proporcionan. Tras el recorte, se reescalía la imagen hasta tener un tamaño de  $256 \times 256$ .

Por otro lado se crean los *Heat-maps* para cada landmark. Para esto se crea una imagen de tamaño  $128 \times 128$  por cada landmark marcando el punto con ayuda de una distribución normal de dos dimensiones centrada en las coordenadas del landmark y usando una desviación típica de  $\sigma = 7$ .

Tras esto se entranan las cuatro *ITLs*. A los datos de entrada se les aplican técnicas de *data-augmentation* como rotaciones, traslaciones, reescalados y occlusiones. Para esta etapa se usa

## 9. Solución propuesta y experimentos realizados

también un optimizador Adam con un *learning rate* de 0.001 y los mismos valores para  $\beta_1$  y  $\beta_2$ .

Finalmente, durante la etapa de *fine-tuning* se establece un *learning-rate* de 0.0001 en las *ITLs* mientras que el del *Encoder* se mantiene en su valor por defecto de  $2 \times 10^{-5}$  y cambiando el valor de  $\beta_1 = 0.9$ .

### 9.1.4. Bases de datos usadas por el framework

Para el entrenamiento no supervisado del *AAE*, se emplearon los siguientes datasets unidos:

- **VGGFace2** : Contiene un total de 3.3 millones de imágenes de rostros en distintas poses, edad, iluminación, etnia, etc... Del dataset eliminaron las imágenes de rostros que tuviera una altura menor a 100 píxeles, quedando un total de 1.8 millones de caras.
- **AffectNet** : Se trata de un dataset de 228 mil imágenes en una gran variedad de poses, iluminación, etc..

En total usaron unas 2.1 millones de imágenes.

Para el entrenamiento supervisado se emplearon los siguientes datasets:

- **300-W** : une diversos datasets de rostros etiquetados con 68landmarks de manera semi-automática como son **LFPW**, **AFW**, **HELEN** y **XM2VTS**. Además de añadir datos propios. En total emplearon 3,148 (aproximadamente el 80 %) imágenes para el entrenamiento y 689 para test, las cuales se dividieron en dos grupos, uno de 554 imágenes considerado el grupo test estándar, y otro de 135 imágenes difíciles.
- **AFLW** : contiene un total de 24,386 imágenes *in-the-wild* con un amplio rango de poses distintas. Se emplearon 20,000 (aproximadamente el 80 %) imágenes para test y 4,386 para entrenamiento. Las imágenes vienen etiquetadas con 21 landmarks, pero en el framework se entrena la red para predecir 19.
- **WFLW**: Es la más reciente de las empleadas y tiene un total de 10,000 imágenes. Se usan 7,500 para entrenamiento y 2,500 para test. Las imágenes tienen un total de 98 landmarks anotados.

Como podemos observar, los conjuntos de datos disponibles para el entrenamiento de redes especializadas en la identificación de landmarks faciales son variados y con muchos ejemplos de entrenamiento. En nuestro caso concreto, no contamos con tantos ejemplos y apenas hay bases de datos forenses con landmarks cefalométricos, lo cual es una dificultad añadida al problema, pues solo podremos entrenar con los datos que se nos proporcionan y como veremos son muy pocos.

## 9.2. Métricas

En esta sección, se van a presentar las principales métricas de error que se emplearán para estudiar la bondad de los resultados que se obtengan en futuros capítulos.

### 9.2.1. Métricas usadas en el entrenamiento

Las métricas que se muestran durante el entrenamiento utilizadas por el modelo son:

### 9.2.1.1. MSE

Se trata del error cuadrático medio, una función empleada típicamente en problemas de regresión y que tiene la siguiente expresión:

$$MSE = \frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2$$

En nuestro caso concreto, este error se calcula entre los mapas de calor de los landmarks reales y predichos a nivel de pixel, es decir  $y_i$  en la fórmula anterior corresponde a cada pixel del heatmap del landmark real y por tanto  $\hat{y}_i$  es el valor del pixel correspondiente en el heatmap predicho. Dichas diferencias se suman en todos los heatmaps y se divide por  $N$  que en este caso es el número total de píxeles juntando todos los Heat-maps.

Este error se empleará para realizar *backpropagation* a través de las ITLs de la red y será el responsable del aprendizaje de la red.

### 9.2.2. Métricas empleadas en validación y testing

#### 9.2.2.1. Error de reconstrucción

El valor del error de reconstrucción en las imágenes de validación nos llevará en la sección de experimentación a realizar hipótesis sobre la relación entre este error y el NME.

Se trata de la función de coste **L1** que se aplica a la imagen original y la reconstruida y nos proporciona una medida del error de reconstrucción a nivel de pixel. Su expresión es la siguiente:

$$\text{Reconstruction Loss} = \frac{\sum_{i=1}^n |p_i - q_i|}{N}$$

Donde  $p_i$  representa cada pixel de la imagen reconstruida por la red,  $q_i$  el equivalente en la imagen real y  $N$  el total de píxeles presentes en la imagen.

#### 9.2.2.2. Normalized Mean Error

Este será el error utilizado para medir la bondad de los resultados. Tiene la siguiente expresión para cada landmark de los 30:

$$NME = \frac{\sqrt{\sum_{i=1}^n (p_{i,1} - q_{i,1})^2 + (p_{i,2} - q_{i,2})^2}}{N}$$

Así, si en total se han identificado en las imágenes de validación un total de  $n$  landmarks de cierto tipo en las imágenes de test o validación, se calcula la distancia  $L_2$  entre el landmark de la imagen  $i$  predicho y el real. Y tras esto se suman todos los valores obtenidos. Como podemos ver, idealmente  $n$  debería ser el total de imágenes que hay en validación o test, pero no todos los landmarks aparecen en todas las imágenes, por lo tanto  $n \leq M$  siendo  $M$  el total de imágenes en validación.

En dicha expresión  $N$  es el término por el que se normaliza, y varía según el problema. En nuestro caso normalizaremos la media del numerador por el ancho de la imagen que es de 256 píxeles, ya que la distancia máxima permitida entre dos landmarks sería esta, y así todos los valores quedarían entre 0 y 1 siendo 0 una estimación perfecta del landmark, y 1 un

## 9. Solución propuesta y experimentos realizados

error total en el marcado. Es importante la normalización debido a la heterogeneidad de los tamaños de los datos de entrada. Así, el NME extraído en cada dato es comparable con los del resto.

### 9.2.2.3. SSIM

Se trata del *structural similarity index* (SSIM) y da una medida de *similaridad* entre dos imágenes [WBSSo4], su expresión es la siguiente:

$$SSIM(x, y) = [l(x, y)]^\alpha [c(x, y)]^\beta [s(x, y)]^\gamma$$

Dónde  $x, y$  son las dos imágenes que van a ser comparadas.

La componente  $c(x, y)$  hace referencia a la función de comparación del contraste de las dos imágenes y viene dada por la siguiente expresión:

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

Dónde  $\sigma_x, \sigma_y$  hacen referencia a la desviación estándar de cada imagen.

La componente  $s(x, y)$  es la función de comparación estructural entre las dos imágenes y viene dada por la siguiente expresión:

$$s(x, y) = \frac{\sigma_{xy} + C/2}{\sigma_x\sigma_y + C/2}$$

Dónde  $\sigma_{xy}$  denota la covarianza.

La componente  $l(x, y)$  hace referencia a la *luminosidad*, pero en el caso de 3FabRec se prescinde de esta componente, además los exponentes  $\alpha, \beta, \gamma$  se igualan a 1.

Por otra parte siguen las indicaciones del paper original de SSIM que recomiendan usar estas comparaciones en regiones de la imagen y promediarlas en vez de aplicarlas sobre todo el conjunto de píxeles de la imagen, es por ello que la expresión final queda:

$$cs(x, y) = \frac{1}{|w|} \sum c(x_w, y_w) s(x_w, y_w)_w$$

Dónde  $w$  representa la ventana sobre la que se aplica la función y  $|w|$  el total de ventanas. En nuestro caso se emplean ventanas de tamaño  $31 \times 31$ .

## 9.3. Preprocesamiento de los datos

### 9.3.1. Identificación de caras en las imágenes

Antes de poder entrenar el modelo, es necesario adecuar el dataset para que pueda usarse modelo. El framework está pensado para recibir como entrada una imagen, a la cual, se le hace un recorte a la región que ocupa el rostro con ayuda de un *bounding box* predefinido y determinado por las coordenadas en la imagen de las cuatro esquinas o bien siguiendo los landmarks descritos por el fichero auxiliar asociado a cada imagen. En nuestro caso, debido a que la mayoría de landmarks son internos y no se suelen situar en los bordes de la cara,

hemos pensado que la mejor manera de entrenar la red es que se realice el recorte de la imagen de entrada de acuerdo a un *bounding box*. Para ello fue necesario emplear una red especializada en la detección de caras en imágenes y que nos proporcionaba como salida una lista con los *bounding boxes* de todas las caras detectadas. La red se denomina **Facenet**, y se puede importar de la librería *facenet* de *pytorch* como *mtcnn*.

Así pues, se realizó un primer estudio para determinar los mejores parámetros para *facenet*. Existen muchos parámetros editables, pero la mayoría se dejaron con sus valores por defecto, únicamente se consideró modificar:

1. **image\_size**: Determina el tamaño de la imagen de salida.
2. **select\_largest**: Si su valor es *true*, se devuelve la cara más grande detectada, y si su valor es *false*, se devuelve la de mayor confianza detectada.

En nuestro caso, primero se probó con un *image\_size* de  $256 \times 256$ , pero esto tenía problemas pues en ocasiones no podían construirse *bounding boxes* de este tamaño y producían error, por lo que se redujo a  $128 \times 128$ . Por otro lado, observando las imágenes del dataset de entrenamiento, nos dimos cuenta que en algunas imágenes aparecían varios sujetos, y en ocasiones la cara más “grande” no pertenecía al sujeto principal. Es por ello que se decidió establecer el parámetro *select\_largest* a *False*, y así obtener como salida una lista de *bounding boxes* ordenados de mayor a menor según el nivel de confianza. No obstante, en algunos casos se tuvo que elegir con cuidado cuál de los *bounding boxes* proporcionados se correspondía con el sujeto principal.

Una vez establecidos los parámetros, se detectaron problemas en la identificación de caras en algunas imágenes. Debido a que todas las imágenes erróneas estaban en escalas de grises nos dimos cuenta de que la red sólo acepta como entrada imágenes en tres canales, por lo que cada imagen en escala de grises de un solo canal tuvimos que replicar tres veces su canal.

Tras esto, volvimos a ver que en algunas imágenes no se detectaban *bounding boxes*. Esto nos llevó a realizar un breve estudio sobre si era buena idea continuar usando esta red o buscar otra. En este estudio clasificamos las imágenes en tres grupos y aplicamos la red a cada uno de ellos por separado:

1. Imágenes de sujetos en posición **frontal**: en total hay 87 imágenes frontales de las cuales se extraen correctamente el 100 % de los *bounding boxes*.
2. Imágenes de sujetos en posición de 3/4: En este caso hay 57 imágenes de las cuales el 100 % de los *bounding boxes* extraídos por la red son correctos.
3. Imágenes de sujetos en posición de **perfil**: En este caso hay un total de 23 imágenes y se clasifican correctamente 20, lo que supone una precisión del 87 %. Además, las imágenes que fallaban estaban en escala de grises y con malas condiciones de calidad e iluminación.

Por lo tanto, debido al bajo número de ejemplos en los que la red falla, se decidió mantener la red **Facenet** para determinar los *bounding boxes* y además se excluyeron los tres ejemplos dónde fallaba del dataset, lo que hizo que en total se usaran 164 de las 167 imágenes originales.

## 9. Solución propuesta y experimentos realizados



Figura 9.9.: Imágenes de ejemplo del dataset con los *bounding boxes* marcados por *Facenet*. Como podemos observar, aunque son correctos, los *bounding boxes* marcados recortan excesivamente los límites del rostro, pudiendo incluso eliminar partes en las que hay landmarks marcados. Las imágenes han sido generadas con *matplotlib*.

Durante el estudio anterior, se observaron otros hechos dignos de mención. En primer lugar se identificaron dos imágenes repetidas, pero se decidió mantenerlas pues los landmarks presentes en una y otra eran diferentes, lo que podía ayudar al entrenamiento. Por otro lado, durante el entrenamiento se vió cómo había una imagen de un determinado sujeto en la que aprecian simultáneamente dos fotos, una de frente y otra de perfil. Sin embargo, al estar los landmarks anotados únicamente en la imagen de frente se tomó el *bounding box* de la imagen frontal desechariendo el otro para el perfil. En otras dos imágenes se pueden ver un conjunto de varias personas, y la red mostraba erróneamente los *bounding boxes* de personas que no eran el sujeto de estudio, por lo que estudiar cuál de todos los *bounding boxes* devueltos pertenecía al sujeto correcto.

### 9.3.2. Creación del fichero annotations en el dataset

Una vez se identificaban correctamente todos los *bounding boxes*, siguiendo como inspiración el dataset AFLW, se generó un fichero denominado *annotations.csv* dentro de la carpeta del proyecto en el que encuentra el dataset. En este fichero se almacenó para cada imagen una

serie de datos:

- El nombre del fichero.
- El índice de la imagen dentro del dataset.
- Una lista con las coordenadas 2D de cada landmark en la imagen (incluidos los no presentes como situados en la posición  $[-1, -1]$ ).
- Una lista denominada máscara, que representa la visibilidad de cada landmark en la imagen. Los landmarks visibles en la imagen tienen asociado el valor 1, mientras que los no visibles tienen asociado el valor 0. Esta máscara será de gran utilidad a la hora de extraer las métricas, pues hay que realizar las medias de acuerdo al número de landmarks visibles de cada tipo, no se puede usar siempre el valor 30.
- Finalmente, se almacena la posición en coordenadas de la esquina inferior izquierda del *bounding box* junto con el ancho y el alto, para que el framework recorte la imagen quedándose con el rostro únicamente.

### 9.3.3. Reajuste de los bounding boxes

Finalmente, como parte del proceso de determinar los *bounding boxes*, se vió que en todos los casos, dichos rectángulos cortaban parte de las caras de los sujetos (partes en las que había landmarks marcados). Es por ello, que se tuvo que realizar una transformación del *bounding box* sugerido para que manteniendo el centro del mismo abarcase una mayor superficie y que contuviera el rostro completo del sujeto.

La transformación consiste en desplazar la esquina superior izquierda del rectángulo una cantidad  $m$  en el eje de ordenadas y abscisas y establecer el resto de esquinas de acuerdo a este parámetro como se muestra en la [Figura 9.10](#).

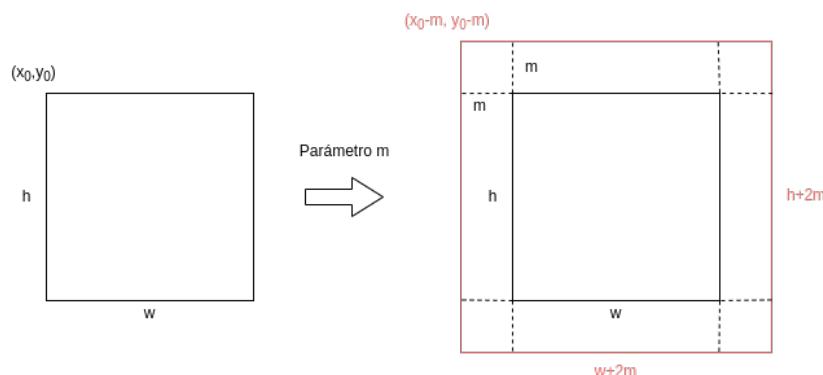


Figura 9.10.: Proceso seguido para la transformación.

No obstante, el framework 3FabRec dispone de un método de recorte y redimensión automática de las caras empleando *bounding boxes*. En un primer momento, forzamos al framework a respetar los *bounding boxes* construidos durante el preprocesamiento, pero los errores de

## 9. Solución propuesta y experimentos realizados



Figura 9.11.: Recorte de las caras tras el reajuste del *bounding box*.

reconstrucción de estas imágenes eran muy elevados. No resultaba práctico para entrenar la red, pues el framework marca los puntos sobre localizaciones que no guardaban parecido con el lugar real del punto en la imagen original. Podemos ver algunos ejemplos de reconstrucción en la [Figura 9.12](#).

Se probó entonces a aplicar al *bounding box* definido, el reajuste que realiza 3FabRec. Este nuevo reajuste dejaba en la mayoría de casos mal marcado el *vertex*, pues suele recortar la parte superior de la cabeza. No obstante los errores de reconstrucción eran mucho menores con esta alternativa y los landmarks predichos se localizaban en las imágenes reconstruidas sobre puntos que guardaban una estrecha relación con la localización real en la imagen original (véase la [Figura 9.13](#)). Se decidió entonces estudiar por qué ocurría esto.

El reajuste que realiza 3FabRec es el siguiente:

1. En primer lugar, se determina el tamaño de la región de interés (ROI, del inglés region of interest) en píxeles. Para ello, el framework por defecto calcula la diagonal del tamaño de las imágenes de entrada (en nuestro caso son  $256 \times 256$ ), y lo redondea al entero superior más próximo (el resultado es de 363 píxeles, en nuestro caso). De este valor se obtienen dos parámetros, el *crop\_size* y el *margin*, de manera que  $crop\_size + margin = roi$  (en nuestro caso  $crop\_size=256$  y  $margin=107$ ).
2. En segundo lugar, se reajusta el bounding box convirtiéndolo siempre en cuadrado de lado  $\max(ancho, alto)$ . Se calcula el factor de escalado  $escala = \frac{crop\_size+margin}{crop\_size}$ , para ampliar el bounding box original uniformemente, de acuerdo al *margin*.

### 9.3. Preprocesamiento de los datos

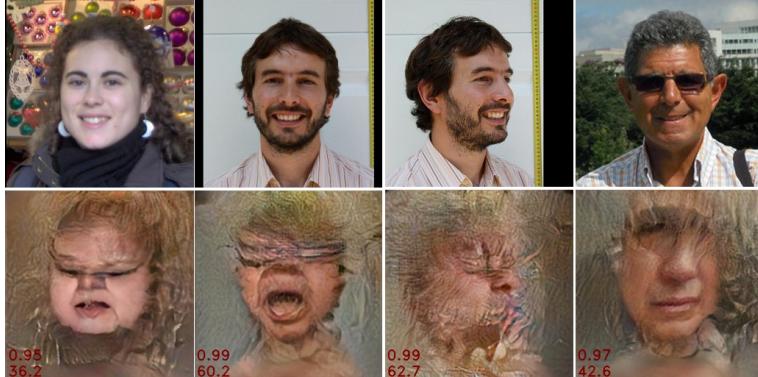


Figura 9.12.: Error estructural y de reconstrucción de las imágenes con el bounding box reajustado. El error de reconstrucción se encuentra en la esquina inferior izquierda, y el estructural encima del anterior. Como podemos observar, en todos los casos son muy elevados y la imagen reconstruida apenas guarda relación con la original.



Figura 9.13.: Error estructural y de reconstrucción de las imágenes con el reajuste del bounding box de 3FabRec. Como podemos ver, sigue habiendo grandes errores de reconstrucción y estructurales, pero se consigue un mejor resultado de la estructura facial. No obstante, el vertex no se aprecia correctamente en las imágenes.

3. Finalmente se reescal a una imagen de tamaño  $256 \times 256$ .

Por otro lado, el entrenamiento no supervisado del AAE hizo que la red aprendiera la distribución de un determinado dataset de imágenes. Estudiando minuciosamente los ficheros encargados de preparar los datos de entrenamiento (`affectnet.py` y `vgg2face.py`), se vio que en ambos, el tamaño de las imágenes es de  $256 \times 256$ , pero no se especifica un valor determinado para el ROI, por lo que este se calcula por el framework de la misma manera que hace con nuestras imágenes, realizando un ajuste de los bounding boxes idéntico al nuestro. Además, para el ajuste fino del encoder utilizando las imágenes de AFLW, de nuevo se aplicaron estas transformaciones.

Esto, puede explicar por qué en el primer caso no se reconstruyen bien las imágenes, ya que se fuerza a que no se apliquen estas transformaciones a las imágenes de entrada. De

## 9. Solución propuesta y experimentos realizados

esta manera, la distribución de las imágenes de entrada cambia radicalmente con respecto a la distribución con la que fue entrenada la red y su posterior *fine-tuning* con AFLW. Sin embargo, si permitimos que 3FabRec realice las transformaciones descritas anteriormente, conseguimos que la distribución de los datos de entrada (pese a seguir siendo diferente) sea lo mas “parecida” posible a la distribución sobre la que fue entrenada la red, mejorando la reconstrucción de las imágenes.

Por lo expuesto anteriormente, se decide continuar con el reajuste que realiza el framework para lograr una mejor reconstrucción, aunque se pierda el marcado correcto del *vertex*.

### 9.3.4. Separación en conjuntos de entrenamiento y validación

Separamos la base de datos proporcionada en conjuntos de entrenamiento, validación y test. Por el limitado número de ejemplos que tenemos, será necesario emplear **validación cruzada** para la elección entre los modelos que se prueben. Así pues, de las **164 imágenes** que usaremos en total, establecemos una semilla para que los procesos aleatorios sean replicables en cualquier máquina y con ayuda de la función *random.shuffle* de python sepáramos los datos en:

- **Conjunto de test:** 33 imágenes (20 % del total de imágenes).
- **Conjunto de entrenamiento:** 131 imágenes (80 % del total de imágenes).

Por otro lado, el conjunto de entrenamiento se subdivide a su vez en cinco subconjuntos, con la idea de realizar **5-fold cross-validation**. Cada subconjunto estará compuesto por **26 imágenes** exceptuando el último que tendrá **27**.

Para esta técnica se realizan cinco ejecuciones independientes en la cual se toman cuatro de los cinco subconjuntos de entrenamiento para entrenar el modelo y se deja el último subconjunto para validar el modelo. Así, en cada iteración se cambia el conjunto que se empleará para validación, con la finalidad de mejorar la calidad de las decisiones que se tomen sobre el rendimiento, pues la decisión estará mejor informada.

Con cada propuesta de modelo, se medirá su rendimiento usando **5-fold cross-validation**, obteniendo 5 salidas de validación para cada modelo. Cada salida se compone de una tabla en la que aparece el NME medio por landmark (teniendo en cuenta sólamente los marcados en validación), y una tabla en la que para cada imagen de validación se obtiene su NME medio (haciendo la media del NME cometido por cada landmark), y el error de reconstrucción. Para la elección del modelo final se tendrá en cuenta:

- Una tabla en la que se promedian las cinco tablas obtenidas con el NME medio por landmark.
- Una tabla en la para cada imagen del conjunto de entrenamiento, se tiene el NME cometido y el error de reconstrucción. Esta tabla se obtiene concatenando todas las predicciones de cada partición de cross-validation.

## 9.4. Experimentación

### 9.4.1. Hipótesis iniciales

Debido a que partimos de **3fabRec**, un framework diseñado y probado de forma exhaustiva por Björn Browatzki y Christian Wallraven, en los experimentos se va a respetar la arquitectura de la red en su totalidad, sin añadir ni eliminar capas. Del mismo modo, debido a que la elección del optimizador **Adam** empleado por los autores ha sido fruto de un proceso de experimentación exhaustiva por parte de los mismos no, se va a probar a cambiarlo. Además, el optimizador Adam es adaptativo y tiene muy buen rendimiento en general, por lo que no tenemos ninguna razón que nos lleve a pensar que lograríamos grandes mejoras cambiando de optimizador.

También, en la creación de los *Heat-maps* se ha optado por mantener el valor de  $\sigma = 7$  que se recomienda en el paper [BW20], ya que de nuevo ha sido probado minuciosamente por los autores del framework.

### 9.4.2. Modelo Base

Buscamos un **modelo base** con el que obtener los primeros resultados, y que iremos refinando hasta obtener el modelo solución. El framework **3FabRec** viene por defecto con cuatro modelos. Todos ellos han sido entrenados en la fase de aprendizaje no supervisado con la unión explicada de los datasets *VGG2Face* y *Affectnet*. Uno de ellos viene sin entrenamiento posterior de landmarks en ningún dataset, y los otros tres vienen con un entrenamiento y *fine-tuning* del encoder en los datasets *300w*, *AFLW* y *WFLW*.

Se ha optado por usar como modelo base el que ha sido entrenado en *AFLW* porque el número de landmarks que predice este modelo es similar al de nuestro problema, ya que predice 21 landmarks, y la estructura de estos landmarks (visibles y no visibles) es similar a la nuestra.

Por otro lado, debido a que no conocemos todavía el efecto que tiene el entrenamiento con la etapa de *ajuste fino* del encoder sobre los datos, vamos a realizar un entrenamiento únicamente de las ITLs durante un total de **60 épocas**, apreciándose la convergencia del *mse* (que recordamos, es la medida de error entre los *Heat-maps* predichos por el modelo y los reales), y no utilizaremos técnicas de *data augmentation*. Por otro lado, los parámetros que se han usado han sido los que sugieren por defecto en el framework, ya que aún no tenemos información suficiente sobre el rendimiento como para cambiarlos. Dichos valores son:

- El *learning rate* para el optimizador Adam usado en el entrenamiento de las ITLs se establece en 0.001.
- El valor del parámetro  $\beta_1$  se establece en 0.9.
- El valor del parámetro  $\beta_2$  se establece en 0.999.

Los resultados obtenidos por imagen en cada partición de **cross validation 5-fold** se pueden encontrar en **Apéndice A**. En estas tablas podemos observar la media del NME obtenido entre todos los landmarks presentes en la imagen, los no marcados no computan. Los resultados obtenidos son en general buenos para ser el modelo base. El NME no es muy elevado, aunque el error de reconstrucción y estructural sí lo son. Podemos concluir así que no se

## 9. Solución propuesta y experimentos realizados

están reconstruyendo fielmente las imágenes a nivel de píxel, aunque como vemos en la [Figura 9.15](#), la reconstrucción de las caras es apropiada en general. No obstante, esto podría afectar al marcado de landmarks, puesto que estos se marcan en primer lugar sobre la imagen reconstruida, y luego se trasladan a la original.

Tabla 9.1.: Media del error NME obtenido por landmark entre todas las particiones de cross-validation. Marcamos en amarillo el vertex pues no está correctamente marcado en todas las imágenes.

	<b>Landmark</b>	<b>Media NME</b>
0	Menton	4.176
1	Gnathion	2.629
2	Pogonion	2.509
3	Prosthion	1.178
4	Labiale Superius	1.965
5	Subnasale	2.056
6	Nasion	2.211
7	Glabella	2.562
8	Vertex	5.904
9	Left Gonion	5.652
10	Right Gonion	5.655
11	Left Zygion	5.351
12	Right Zygion	6.504
13	Left Alare	1.972
14	Right Alare	2.27
15	Left Endocanthion	1.675
16	Right Endocanthion	1.731
17	Left Exocanthion	1.7
18	Right Exocanthion	1.651
19	Left Tragion	4.15
20	Right Tragion	5.639
21	Labiale inferius	1.734
22	Trichion	4.445
23	Supramentale	1.933
24	Left Frontotemporale	2.37
25	Right Frontotemporale	2.97
26	Left Frontozygomaticus	1.566
27	Right Frontozygomaticus	2.483
28	Left Midsurpaorbital	1.168
29	Right Midsurpaorbital	1.429

Los resultados por landmark se pueden ver en la [Tabla 9.1](#) y son buenos en general para tratarse del modelo base, siendo el landmark mejor marcado en media el *Prosthion*, y el peor el *Right Gonion*. Si recordamos la [Figura 5.4](#), sorprende el hecho de que el *Prosthion* es de los landmarks que menos presencia tiene en las imágenes (aparece en menos de 40 imágenes), y sin embargo es de los que mejor aprende a marcar la red. Por otro lado, el *Right Gonion*, pese a aparecer en unas 100 imágenes, resulta más difícil para la red predecirlo. Esto puede deberse a que las imágenes con mayor error de reconstrucción son las de perfil, y es en este

#### 9.4. Experimentación

tipo de imágenes en las que mejor se aprecia dicho landmark.

Por otro lado, podemos ver en la [Figura 9.14](#) las curvas de aprendizaje de cada partición. Todas tiene una gran capacidad de generalización, pues como vemos el error de validación desciende a la par que el de entrenamiento y está poco por encima de este último.

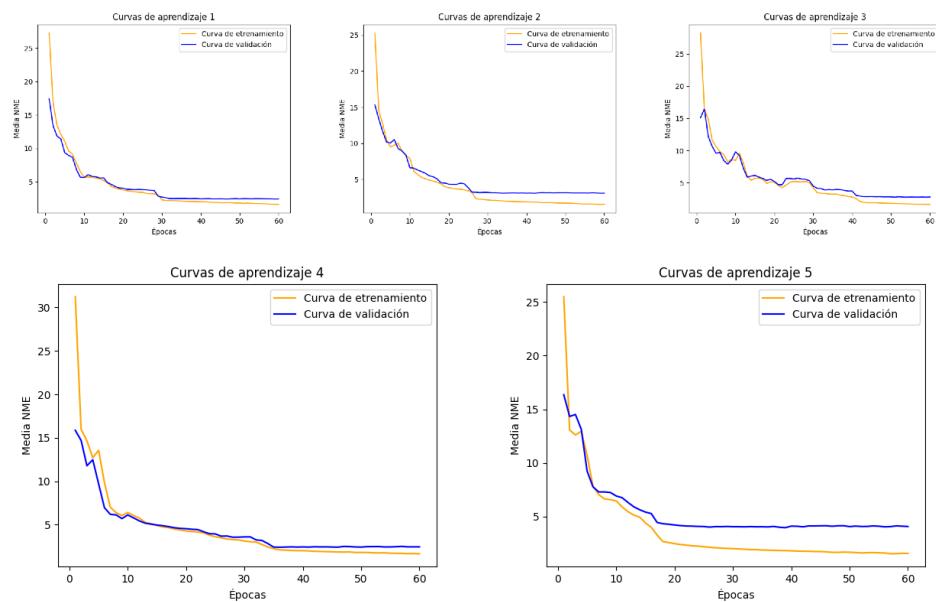


Figura 9.14.: Curvas de aprendizaje en cada partición del modelo base.

En resumen, los resultados parecen buenos para tratarse de una primera aproximación. Sin embargo, consideramos que debido a la gran variabilidad del dataset tanto en posturas, como en calidad e iluminación de imagen, la reconstrucción de las imágenes se ha resentido, especialmente en las imágenes de perfil. Esto provoca que el modelo también marque peor los landmarks, por lo que vamos a intentar mejorar esto en los próximos modelos. Podemos ver el rendimiento del modelo en algunas imágenes en la [Figura 9.15](#).

## 9. Solución propuesta y experimentos realizados

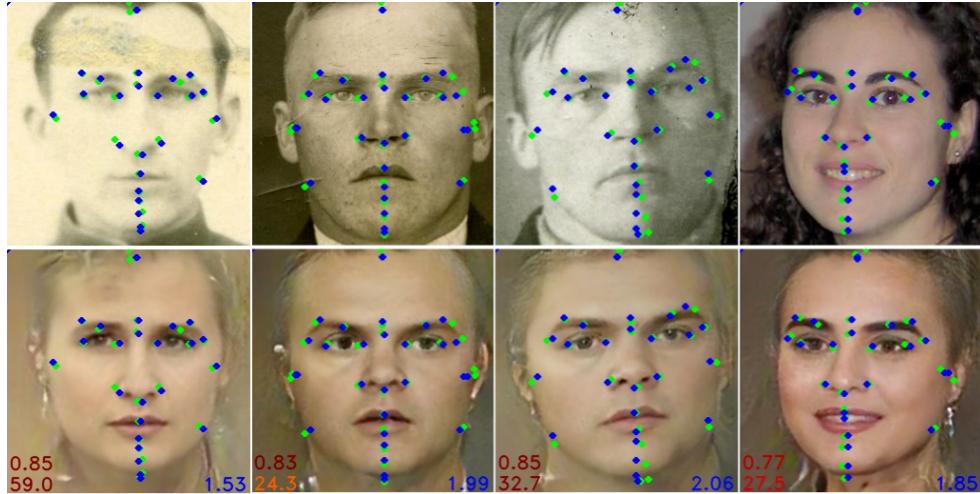


Figura 9.15.: Imágenes pertenecientes a distintos conjuntos de validación para el **modelo base**. Las imágenes de la fila superior son las imágenes reales, y las de la fila inferior las reconstruidas por la red. El marcado se realiza sobre las reconstruidas y luego se lleva a las superiores. Podemos apreciar en verde los landmarks reales y en azul los predichos. El valor que aparece en la esquina inferior derecha es el NME global de la imagen (la media de los NMEs de cada landmark) y en la esquina inferior izquierda aparece el error de reconstrucción y encima el SSIM.

### 9.4.3. Modelo con reentrenamiento del encoder

A la luz de los resultados anteriores, vamos a cambiar el modelo de manera que ahora junto con las ITLs (que seguirán usando el mismo optimizador con los mismos parámetros por defecto), vamos a entrenar los pesos del *encoder*. Con este enfoque queremos enseñar a la red a **codificar mejor** la información presente en las imágenes del dataset (que, son más difíciles que las de los datasets empleados anteriormente), a la par que la enseñamos a marcar landmarks en las mismas. Remarcamos, que la función de coste que se empleará tanto para el entrenamiento de los pesos de las ITLs como los del encoder será el *MSE* entre los mapas de calor predichos y los reales.

Es importante aclarar que este proceso no produce overfitting, pues durante las etapas de entrenamiento los pesos del *decoder* permanecerán congelados, y serán únicamente los pesos del *encoder* los que se actualicen con backpropagation. El optimizador empleado para el entrenamiento del *encoder* es Adam, el mismo que ellos emplean y los parámetros se mantienen por defecto ( $\beta_1 = 0.9$  y  $\beta_2 = 0.999$ ) a excepción del *learning rate*, que se establece a  $2^{-6}$ , en vez de  $2^{-5}$ . El motivo por el cual se realiza este cambio es porque experimentando en esta fase se vio que con el antiguo *Learning rate* no se conseguía mejorar apenas la reconstrucción en validación, pues durante el entrenamiento el error de reconstrucción pronto quedaba estancado en un valor. Para evitar esto se optó por reducir el learning rate consiguiendo mejorar un poco más el error de reconstrucción. El entrenamiento duró en total **100 épocas**, pues como veremos, el modelo mejora más lentamente en el reconocimiento de landmarks. Como podemos observar en la **Tabla 9.2**, los valores son muy similares a los del modelo base, pese a que este ha entrenado durante más épocas. El *Prosthion* sigue siendo el landmark mejor

Tabla 9.2.: Tabla comparativa entre los dos modelos explorados por ahora. Medimos el NME medio a nivel de landmark. En verde se resalta el mejor valor de marcado para cada landmark.

Landmark	M. Base	M. Encoder
Menton	4.176	3.997
Gnathion	2.629	2.606
Pogonion	2.509	2.546
Prosthion	1.178	1.048
Labiale Superius	1.965	2.04
Subnasale	2.056	1.992
Nasion	2.211	2.099
Glabella	2.562	2.736
Vertex	5.904	5.89
Left Gonion	5.652	5.789
Right Gonion	5.655	5.301
Left Zygion	5.351	5.288
Right Zygion	6.504	6.291
Left Alare	1.972	1.959
Right Alare	2.27	2.232
Left Endocanthion	1.675	1.71
Right Endocanthion	1.731	1.647
Left Exocanthion	1.7	1.733
Right Exocanthion	1.651	1.746
Left Tragion	4.15	3.633
Right Tragion	5.639	5.252
Labiale inferius	1.734	1.688
Trichion	4.445	4.762
Supramentale	1.933	1.991
Left Frontotemporale	2.37	2.452
Right Frontotemporale	2.97	3.282
Left Frontozygomaticus	1.566	1.499
Right Frontozygomaticus	2.483	2.516
Left Midsurpaorbital	1.168	1.178
Right Midsurpaorbital	1.429	1.348

marcado, con un valor del error NME ligeramente superior al del modelo base. Sin embargo la verdadera mejora se aprecia en el marcado de landmarks de los perfiles, al mejorar la reconstrucción de los mismos, el *Right Gonion* ha bajado su error a la mitad, y el *Left Gonion* ha reducido su error en dos puntos con respecto al modelo base.

Por otro lado, como podemos ver en las tablas de validación por imágenes en el [Apéndice B](#), el error de reconstrucción ha mejorado muy ligeramente en las imágenes de validación, y no supone una mejora considerable con respecto al modelo base, ocurriendo lo mismo con el error estructural.

Finalmente podemos observar las curvas de aprendizaje para este modelo en la [Figura 9.16](#). A diferencia del modelo anterior, comienza a bajar la capacidad de generalización a partir de la época 20 aproximadamente. El error de entrenamiento comienza a descender muy

## 9. Solución propuesta y experimentos realizados

lentamente con cada época mientras que el de validación se mantiene constante.

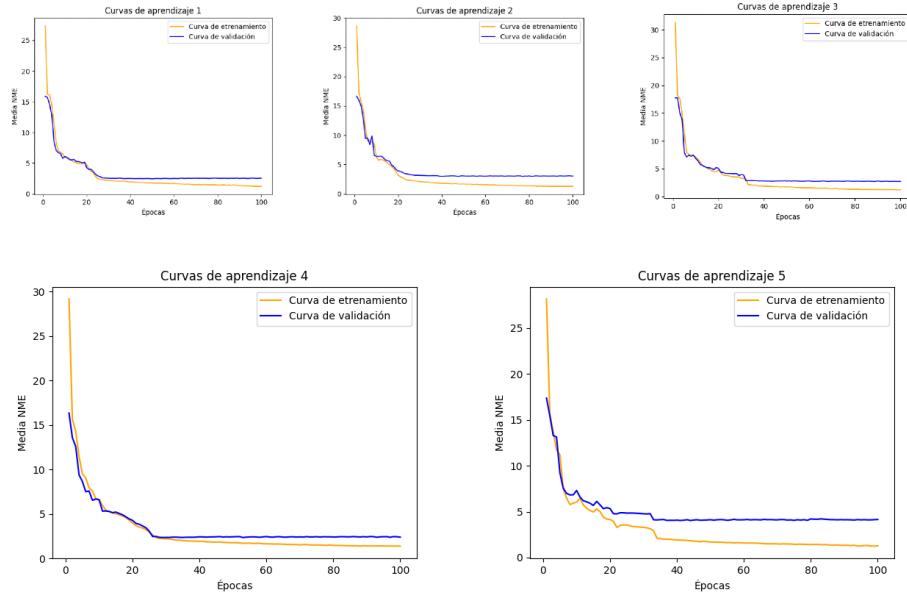


Figura 9.16.: Curvas de aprendizaje en cada partición del modelo de finetuning del encoder.

Viendo estos resultados no parece muy prometedora esta línea de investigación. Bien es cierto que hemos mejorado considerablemente el error de los dos landmarks peor marcados, pero el coste computacional que ha supuesto no compensa para la leve mejora global del modelo. Es por esto por lo que vamos a cambiar el enfoque sobre el *fine-tuning* en el siguiente modelo. Podemos ver el comportamiento del modelo en algunas imágenes de validación de diversas particiones de *cross-validation* en la Figura 9.17

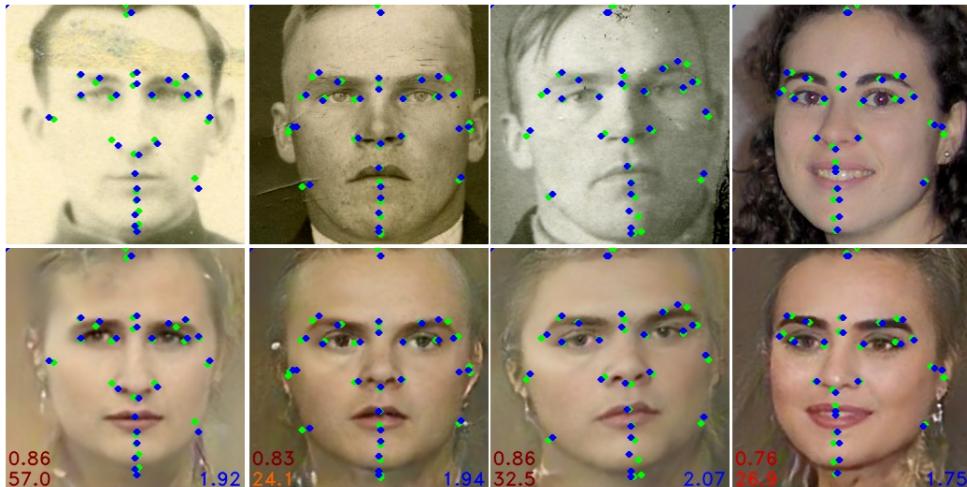


Figura 9.17.: Imágenes pertenecientes a distintos conjuntos de validación para el **modelo de ajuste fino del encoder**.

#### 9.4.4. Modelo con reentrenamiento del decoder

Del mismo modo en que se experimentó con el *encoder* se hizo con el *decoder*. En este enfoque asumimos que el framework sabe codificar correctamente las imágenes pero que no realiza lo suficientemente bien la reconstrucción de las mismas debido a la alta variabilidad del dataset. Esta etapa es completamente análoga a la del modelo anterior (incluyendo el mismo tipo de optimizadores, parámetros y regiones que permanecen congeladas de la red), pero actualizando los pesos del *decoder* y las ITLs mientras se congelan los pesos del *encoder*.

De nuevo los errores por landmark que se muestran en la tabla [Tabla 9.3](#) es muy similar a la de los dos modelos anteriores, por lo que no hemos logrado una gran mejora con este modelo tampoco. Lo único digno de mención es que de nuevo los landmarks que son más visibles en imágenes de perfil han reducido su error debido a la leve mejora de la reconstrucción de las imágenes, aunque esta mejora es prácticamente la misma que en el modelo anterior de ajuste fino del encoder.

Por otro lado, en lo que respecta a los errores promedio de todos los landmarks por imagen, las tablas por partición se pueden consultar en [Apéndice C](#). Aunque los resultados son muy similares a los de los modelos anteriores, el error de reconstrucción de nuevo no mejora apenas, igual que el estructural y los errores NMEs promedio son muy parecidos a los de los otros modelos.

Las curvas de aprendizaje se pueden ver en la [Figura 9.18](#), y vuelven a mostrar un ligero overfitting a partir de la época 30-40, en la mayoría de casos.

## 9. Solución propuesta y experimentos realizados

Tabla 9.3.: Tabla comparativa entre los tres modelos probados. En este caso no se aprecia ninguna mejora considerable con respecto a las introducidas por el modelo de ajuste fino del encoder.

Landmark	M. Base	M. Encoder	M. Decoder
Menton	4.176	3.997	3.991
Gnathion	2.629	2.606	2.634
Pogonion	2.509	2.546	2.585
Prosthion	1.178	1.048	0.97
Labiale Superius	1.965	2.04	2.059
Subnasale	2.056	1.992	2.089
Nasion	2.211	2.099	2.275
Glabella	2.562	2.736	2.912
Vertex	5.904	5.89	6.137
Left Gonion	5.652	5.789	5.665
Right Gonion	5.655	5.301	5.468
Left Zygion	5.351	5.288	5.29
Right Zygion	6.504	6.291	6.14
Left Alare	1.972	1.959	2.051
Right Alare	2.27	2.232	2.295
Left Endocanthion	1.675	1.71	1.827
Right Endocanthion	1.731	1.647	1.61
Left Exocanthion	1.7	1.733	1.809
Right Exocanthion	1.651	1.746	1.734
Left Tragion	4.15	3.633	4.163
Right Tragion	5.639	5.252	4.791
Labiale inferius	1.734	1.688	1.627
Trichion	4.445	4.762	5.196
Supramentale	1.933	1.991	1.839
Left Frontotemporale	2.37	2.452	2.474
Right Frontotemporale	2.97	3.282	3.009
Left Frontozygomaticus	1.566	1.499	1.525
Right Frontozygomaticus	2.483	2.516	2.395
Left Midsurpaorbital	1.168	1.178	1.191
Right Midsurpaorbital	1.429	1.348	1.48

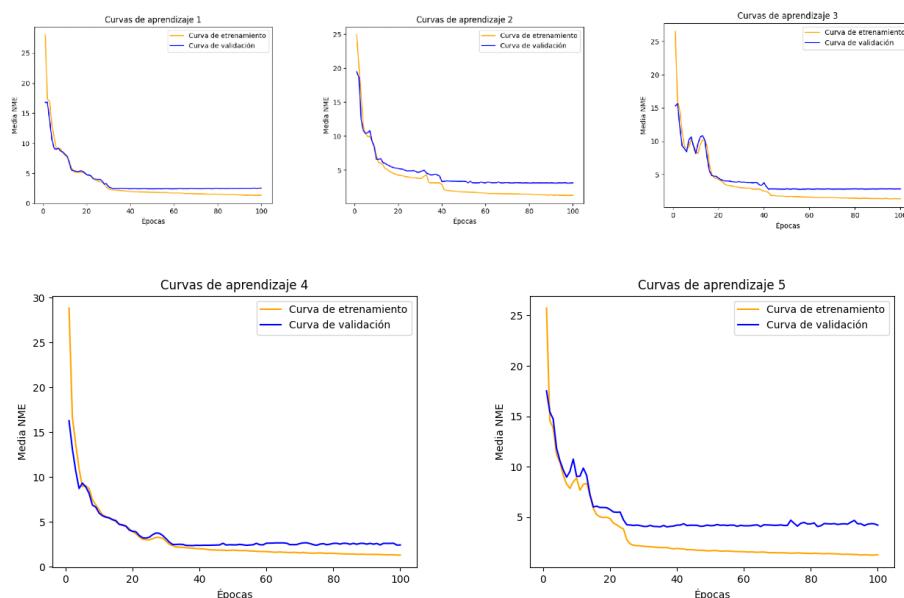


Figura 9.18.: Curvas de aprendizaje en cada partición del modelo de finetuning del encoder.

Tras los resultados obtenidos se ha optado por abandonar también esta línea de investigación pues se obtienen resultados muy similares al modelo base y se debe entrenar por mucho más tiempo (100 épocas frente a 40 del modelo base). Los resultados del modelo sobre algunas imágenes pueden verse en [Figura 9.19](#)

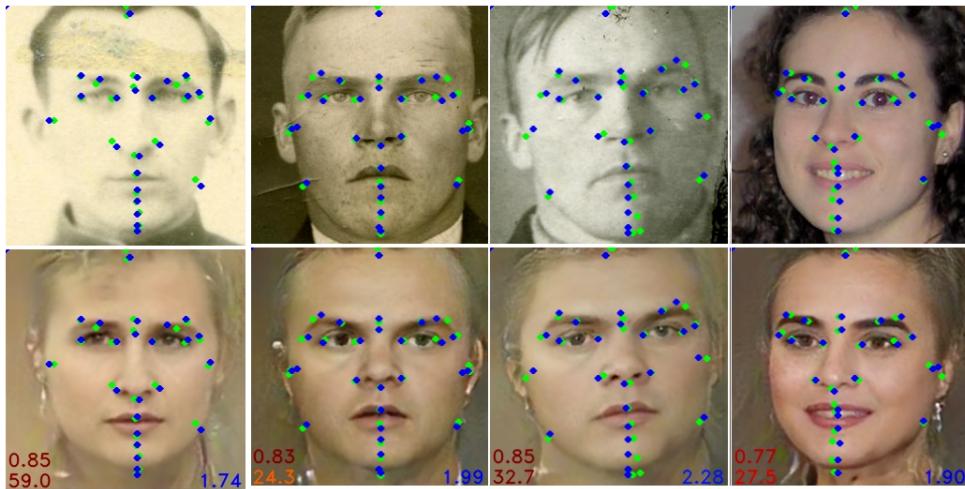


Figura 9.19.: Imágenes pertenecientes a distintos conjuntos de validación para el **modelo de ajuste fino del decoder**.

#### 9.4.5. Modelo con Data Augmentation

Vamos a optar a continuación por un nuevo enfoque. En lugar de mejorar el error de reconstrucción vamos a centrarnos en mejorar el marcado de landmarks en las imágenes reconstruidas que genera el modelo preentrenado. Como pudimos ver en el modelo base, se consiguieron buenos resultados del *NME* pese a haber entrenado sin realizar técnicas de *data augmentation*. Es por ello que en este segundo experimento vamos a realizar los siguientes cambios en el entrenamiento:

- En primer lugar se realizará un entrenamiento del modelo similar al del modelo base durante 40 épocas.
- Tras esto, se entrenará el modelo con el dataset al cual se le aplicarán técnicas de data augmentation, para aumentar el número de datos de entrenamiento e incrementar aún más la variabilidad de los mismos, de manera que cada imagen del dataset sufrirá de manera aleatoria una traslación ( $\pm 4\%$ ), un reescalado ( $\pm 5\%$ ), una rotación ( $\pm 30$  grados) y occlusiones parciales. El entrenamiento en este nuevo dataset será durante 40 épocas tras acabar las de la etapa anterior.
- En total serán 80 épocas de entrenamiento sumando las dos etapas anteriores.

De nuevo, el resto de parámetros continuarán con los valores por defecto de la etapa anterior, pues los resultados obtenidos han sido buenos hasta el momento y nada sugiere tener que cambiarlos.

## 9. Solución propuesta y experimentos realizados

Tabla 9.4.: Tabla comparativa entre todos los modelos probados.

Landmark	M. Base	M. Encoder	M. Decoder	M. Data Augmentation
Menton	4.176	3.997	3.991	3.006
Gnathion	2.629	2.606	2.634	1.497
Pogonion	2.509	2.546	2.585	1.359
Prosthion	1.178	1.048	0.97	0.815
Labiale Superius	1.965	2.04	2.059	1.531
Subnasale	2.056	1.992	2.089	1.272
Nasion	2.211	2.099	2.275	1.241
Glabella	2.562	2.736	2.912	1.545
Vertex	5.904	5.89	6.137	3.99
Left Gonion	5.652	5.789	5.665	1.846
Right Gonion	5.655	5.301	5.468	1.673
Left Zygion	5.351	5.288	5.29	3.322
Right Zygion	6.504	6.291	6.14	4.441
Left Alare	1.972	1.959	2.051	1.585
Right Alare	2.27	2.232	2.295	1.362
Left Endocanthion	1.675	1.71	1.827	1.227
Right Endocanthion	1.731	1.647	1.61	1.129
Left Exocanthion	1.7	1.733	1.809	1.186
Right Exocanthion	1.651	1.746	1.734	1.12
Left Tragion	4.15	3.633	4.163	2.066
Right Tragion	5.639	5.252	4.791	2.234
Labiale inferius	1.734	1.688	1.627	1.032
Trichion	4.445	4.762	5.196	3.37
Supramentale	1.933	1.991	1.839	1.109
Left Frontotemporale	2.37	2.452	2.474	1.386
Right Frontotemporale	2.97	3.282	3.009	1.826
Left Frontozygomaticus	1.566	1.499	1.525	1.036
Right Frontozygomaticus	2.483	2.516	2.395	1.057
Left Midsurpaorbital	1.168	1.178	1.191	0.854
Right Midsupraorbital	1.429	1.348	1.48	0.945

Como podemos ver en la [Tabla 9.4](#), los errores se han reducido en todos los landmarks. Además, la red ha conseguido aprender a marcar con un alto grado de precisión los puntos que se ven en el perfil de la cara también, siendo esta alternativa menos costosa que la de los dos modelos anteriores tanto en número de épocas como en parámetros que se actualizan de la red (en este nuevo modelo sólo se actualizan los pesos de las ITLs). El punto mejor marcado sería el *prosthion* de nuevo, y el peor marcado el *right zygion*. Sin embargo, la diferencia en términos de error entre ambos es mucho menor que en el caso del modelo base.

En lo que se refiere al error por imagen, como podemos ver en el [Apéndice D](#), el error desciende mucho en comparación con los modelos anteriores pese a que el de reconstrucción continua elevado.

Finalmente las curvas de aprendizaje obtenidas se pueden observar en [Figura 9.20](#). Como vemos presentan una gran capacidad de generalización, pues el error de entrenamiento y validación decrecen de forma continua y muy próxima. Algunas imágenes de ejemplo

## 9.4. Experimentación

podemos verlas en la Figura 9.21.

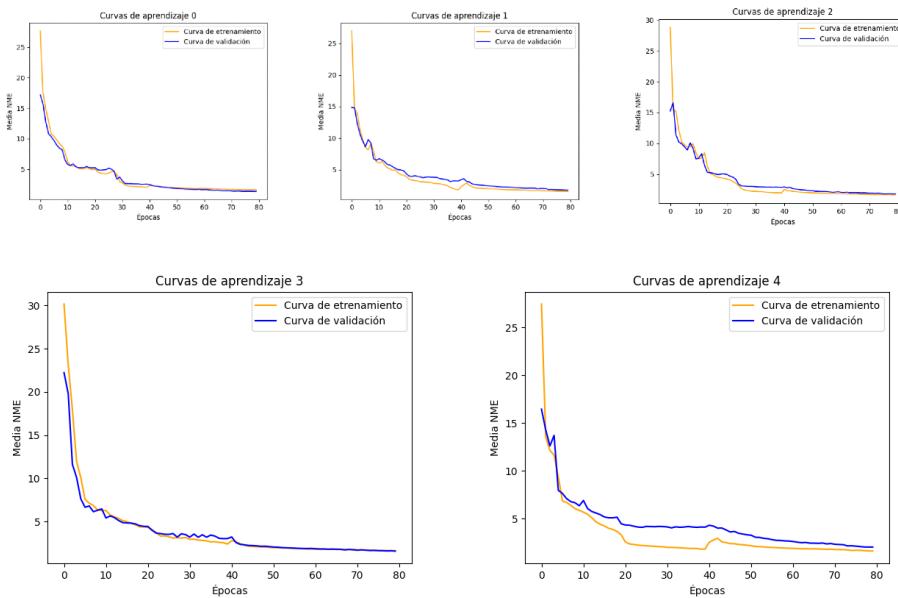


Figura 9.20.: Curvas de aprendizaje en cada partición del modelo de data augmentation.

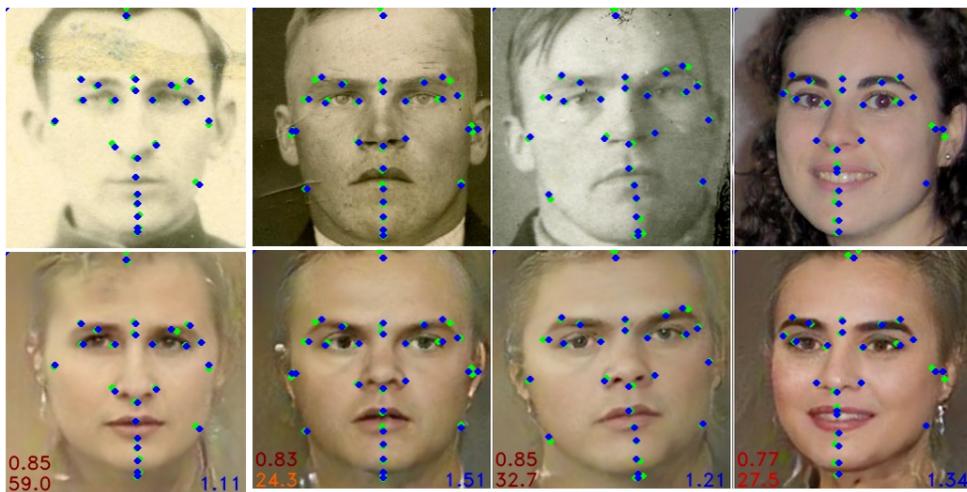


Figura 9.21.: Imágenes pertenecientes a distintos conjuntos de validación para el **modelo data augmentation**.

### 9.4.6. Elección de modelo y obtención de resultados

Una vez analizados los distintos modelos que se proponen para resolver el problema, vemos que la alternativa más prometedora resulta ser la última, la de aplicar técnicas de *data*

## 9. Solución propuesta y experimentos realizados

*augmentation* al dataset. Es por ello que ahora tomaremos el conjunto de entrenamiento en su totalidad (sin realizar particiones como hicimos para *cross validation*) y vamos a usarlo para entrenar el modelo durante las mismas épocas que en la sección anterior y con los mismos parámetros y transformaciones. Posteriormente usaremos el conjunto de test de 33 imágenes que separamos al principio de esta sección, y que hasta ahora no hemos visto para evitar *data snooping*, para validar el modelo.

Tabla 9.5.: Tabla comparativa entre los resultados del modelo de data augmentation en el conjunto de validación y test

Landmark	NME Test	NME Validación
Menton	0.946	3.006
Gnathion	3.323	1.497
Pogonion	3.149	1.359
Prosthion	0.89	0.815
Labiale Superius	1.134	1.531
Subnasale	2.982	1.272
Nasion	3.1	1.241
Glabella	3.089	1.545
Vertex	6.568	3.99
Left Gonion	4.334	1.846
Right Gonion	1.297	1.673
Left Zygion	4.813	3.322
Right Zygion	4.465	4.441
Left Alare	3.531	1.585
Right Alare	3.641	1.362
Left Endocanthion	3.917	1.227
Right Endocanthion	4.685	1.129
Left Exocanthion	3.936	1.186
Right Exocanthion	4.526	1.12
Left Tragion	2.365	2.066
Right Tragion	1.678	2.234
Labiale inferius	1.083	1.032
Trichion	1.731	3.37
Supramentale	1.064	1.109
Left Frontotemporale	1.208	1.386
Right Frontotemporale	1.282	1.826
Left Frontozygomaticus	1.024	1.036
Right Frontozygomaticus	0.943	1.057
Left Midsurpaorbital	0.846	0.854
Right Midsupraorbital	2.027	0.945

En primer lugar, como se puede apreciar en la [Tabla 9.5](#) obtenemos:

- En general los landmarks empeoran ligeramente su error NME. Esto es normal pues en la etapa anterior todas las decisiones para mejorar los modelos se tomaron en función de los resultados de validación. Sin embargo estos datos nuevos en los que se está probando el modelos no han sido estudiados previamente, por lo que es normal

que baje el rendimiento. No obstante, los resultados siguen siendo buenos, y no se empeoran excesivamente en la mayoría de casos.

- En total empeoran su error 19 de los 30 landmarks marcados. En algunos casos el error se duplica, como puede ser el *Gnation* y en otros empeora levemente, como ocurre con el *Left Zygion*.
- En total mejoran su error 11 de los 30 landmarks, destacando la importante mejora en la identificación del *Menton*, y algunas más leves como las del *Right/Left Frontotemporale*.

Por otro lado, las curvas de aprendizaje durante el entrenamiento del modelo se pueden observar en la [Figura 9.22](#). En ellas podemos observar una mayor distancia entre la curva de entrenamiento y validación, fruto de nuevo de no haber usado el conjunto de test para tomar decisiones, como si que hicimos con los datos de validación en *cross-validation*. Podemos ver el rendimiento del modelo en algunas imágenes en [Figura 9.23](#).

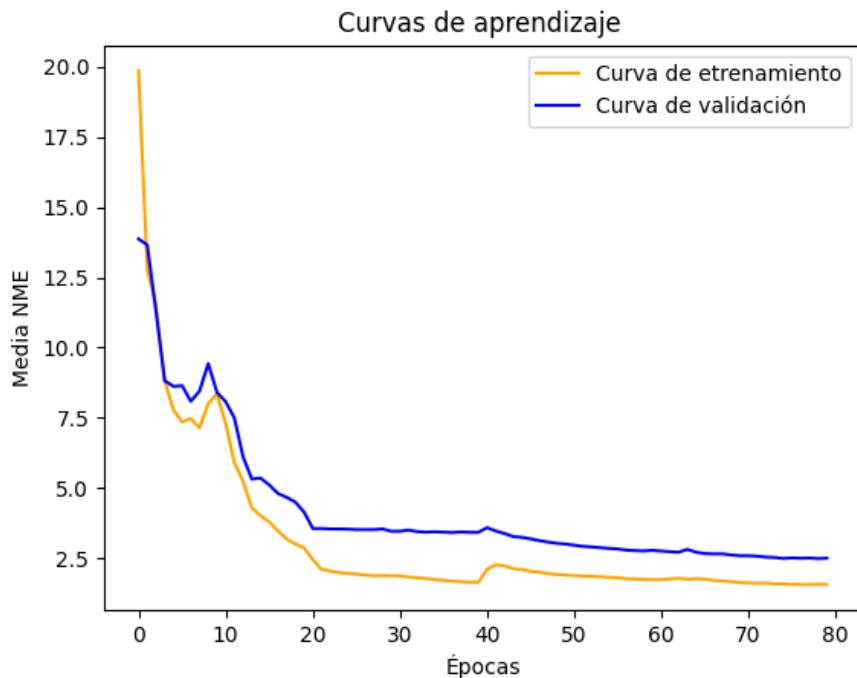


Figura 9.22.: Curvas de aprendizaje durante el entrenamiento del modelo de data augmentation validado en el conjunto de test.

## 9. Solución propuesta y experimentos realizados

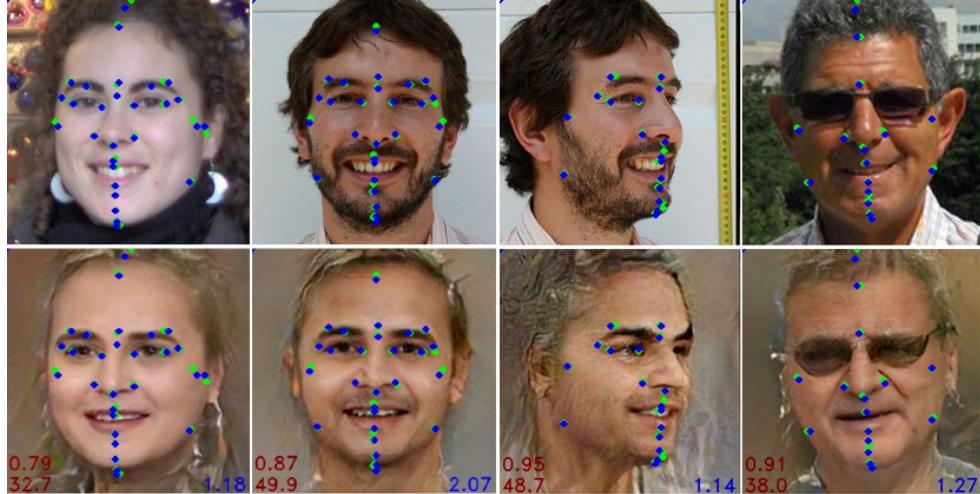


Figura 9.23.: Rendimiento del modelo final elegido con data augmentation en imágenes del conjunto de test.

Como podemos ver, en general el modelo se comporta bien en la tarea del reconocimiento de landmarks, es capaz de predecirlos con una tasa de error NME muy pequeña, por lo que podemos concluir que el modelo resuelve la tarea que nos habíamos propuesto con éxito. No obstante, el éxito real del modelo debe supervisarlo un experto forense, pues no sabemos si los errores cometidos en la identificación de landmarks son errores asumibles o si se requiere un mayor grado de precisión.

## 9.5. Comparativa entre 3FabRec y HyperFace-ResNet101

Vamos a comparar ahora el modelo final obtenido a partir de **3FabRec** con otro que fue diseñado para el mismo propósito basado en **HyperFace-Resnet101**, otra red especializada en el marcado de landmarks, realizado por Guillermo Gómez.

### 9.5.1. Preprocesamiento y base de datos empleada

En lo que respecta al preprocesamiento de los datos, nuestro modelo basado en **3FabRec** únicamente emplea como etapa de preprocesamiento la identificación de bounding boxes en el conjunto de datos inicial de cara a poder entrenar la red. Por otro lado, este estudio se limita a usar las 167 imágenes proporcionadas.

En cambio, el modelo basado en **HF-ResNet** además de emplear también un detector de caras para los bounding boxes, contaba con un dataset auxiliar de modelos 3D de personas sobre las que estaban marcados 27 de los 30 landmarks que se predicen. Lo cual sirvió para aumentar el conjunto de datos de entrenamiento realizando proyecciones 2D a partir de estos modelos.

### 9.5.2. Métricas empleadas

La métrica empleada en el trabajo donde se presenta el modelo es el **RMSE**. Se trata de la raíz del error cuadrático medio y su expresión es la siguiente:

$$RMSE(y, \hat{y}) = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - \hat{y}_i)^2}$$

donde  $y_i$  representa las coordenadas del landmark real y  $\hat{y}_i$  el predicho homólogo número  $i$  del total de  $N$  landmarks marcados.

Para poder realizar una comparación entre ambos métodos, se extrae esta métrica del rendimiento del modelo final en el conjunto de test.

### 9.5.3. Comparación de resultados

En primer lugar comparamos ambos modelos calculando la mediana del *RMSE* producido en cada landmark, así como los percentiles 25, 50 y 75. Podemos ver estos resultados en la [Tabla 9.6](#).

Tabla 9.6.: Tabla comparativa a nivel global entre los dos modelos. Como podemos observar, el modelo basado en **HyperFace-Resnet101** obtiene mejores resultados en global en todos los campos.

Modelo	RMSE	P25	P50	P75
Modelo HyperFace-Resnet101	3.4106	1.802	2.7569	4.2849
Modelo 3FabRec	5.3862	3.0991	5.3862	5.9819

A nivel general, parece que el modelo basado en *HyperFace-ResNet101* es mejor que el presentado en este trabajo, no obstante vamos a realizar una comparativa a nivel de landmark entre los dos modelos. Podemos ver esta comparativa en la [Tabla 9.7](#).

- En total 11 landmarks se marcan con mayor precisión con el modelo basado en **3FabRec**.
- En total 19 landmarks se marcan mejor en con el modelo basado en **HyperFace-ResNet101**.

Como podemos observar, en su mayoría los landmarks que el modelo basado en *HyperFace-ResNet101* son más precisos que los marcados por nuestro modelo en una media de 2 píxeles de diferencia.

9. Solución propuesta y experimentos realizados

Tabla 9.7.: Tabla comparativa a nivel de landmarks entre el modelo final basado en 3FabRec y el de HyperFace-REsNet101.

Landmark	RMSE 3FabRec	RMSE HyperFace-Resnet101
Menton	6.43	4.97
Gnathion	5.5	3.84
Pogonion	5.5	3.99
Prosthion	1.73	3.23
Labiale Superius	3.35	2.35
Subnasale	4.9	3.33
Nasion	4.91	3.12
Glabella	5.36	3.89
Vertex	10.95	8.83
Left Gonion	9.7	7.12
Right Gonion	7.23	6.11
Left Zygion	11.49	5.78
Right Zygion	13.47	6.94
Left Alare	5.6	3.50
Right Alare	5.68	2.84
Left Endocanthion	5.11	2.40
Right Endocanthion	6.0	2.32
Left Exocanthion	6.11	3.73
Right Exocanthion	5.92	3.62
Left Tragion	5.34	7.01
Right Tragion	5.81	6.41
Labiale inferius	2.43	3.11
Trichion	5.41	7.02
Supramentale	2.29	3.43
Left Frontotemporale	3.16	3.78
Right Frontotemporale	3.08	3.27
Left Frontozygomaticus	2.25	2.75
Right Frontozygomaticus	2.2	2.80
Left Midsurpaorbital	1.91	2.57
Right Midsurpaorbital	3.05	2.24

## 10. Conclusiones y Trabajos Futuros

La elaboración de este TFG ha sido un proceso de constante cambio y aprendizaje. Al comienzo constó de un periodo de asimilación de conceptos clave, familiarización con las CNN y con la arquitectura de red que se emplearía: el Adversarial Autoencoder. Tras esto comenzó una etapa de estudio minucioso del artículo donde se presentaba 3FabRec, a fin de comprender su funcionamiento a la perfección. Durante este periodo también se investigó acerca de los *landmarks* faciales y su utilización en la resolución de problemas actuales de identificación de personas. Después se profundizó en el problema del marcado de *landmarks*cefalométricos, entendiendo la diferencia entre estos y los faciales, investigando el estado del arte y aprendiendo su utilidad en procedimientos forenses. Finalmente, se implementaron todas las técnicas presentes en las distintas etapas de la experimentación: el preprocesamiento de los datos, el entrenamiento usando *cross-validation 5 fold*, las técnicas de data augmentation y el ajuste fino del encoder y decoder. Tras esto se obtuvo un modelo final que obtenía en validación un NME promedio de todos los landmarks de 1.77 y en el conjunto de test de 2.65.

Los resultados obtenidos con el modelo presentado son prometedores y, aunque se encuentran por debajo en media con respecto al modelo de *HF-ResNet*, cabe destacar el hecho de que el dataset empleado para esta tarea se ha ceñido únicamente a las 167 proporcionadas, sin añadir ejemplos, como los modelos 3D que se emplearon en el otro trabajo para aumentar el dataset, a excepción de técnicas de *data augmentation* que únicamente duplicaron el conjunto de datos de entrenamiento. La mediana del RMSE cometido por nuestro modelo es de 5.3862 píxeles de error, frente a los 3.4106 del modelo basado en *HyperFace*. Podemos así concluir que esta alternativa de *few-shot learning* es prometedora, pues con muchos menos ejemplos de entrenamiento se consiguen resultados competentes. Además resuelve un problema importante y que como hemos explicado lleva mucho tiempo para el experto forense. Con este método el experto tendría que revisar los landmarks predichos para evitar errores, pero ahorraría tiempo.

### 10.1. Objetivos Satisfechos

Todos los objetivos que se habían propuesto al comienzo de este trabajo se han visto realizados con éxito:

1. Se ha realizado una minuciosa investigación sobre el **estado del arte** en el campo concluyendo que estamos en una vía de estudio de la que apenas hay artículos publicados. Además, hemos resumido y estudiado todas las propuestas de los pocos trabajos encontrados relacionados con el problema de forma directa, con el fin de conocer qué vías se han explorado para su resolución con anterioridad.
2. Hemos realizado un estudio para observar la **evolución** de los Autoencoders y las redes Adversarias, comenzando por describir lo que es un Autoencoder clásico, las novedades

## 10. Conclusiones y Trabajos Futuros

que incorporan las GANs, la aparición de los VAE, y finalmente la combinación de ambos en la arquitectura del AAE.

3. Se ha realizado un **estudio de la base de datos** proporcionada viendo la frecuencia de aparición de landmarks en cada imagen, así como el rendimiento del detector de caras sobre las imágenes del dataset agrupadas por frontales, de perfil o 3/4. Con esto se pudo hacer una estimación de los landmarks que serían más difíciles de predecir y se descartaron tres imágenes debido a que el detector de caras no pudo reconocer ninguna en las mismas. Con estos resultados, se construyó un **nuevo dataset** compuesto por las imágenes usadas finalmente junto con los *bounding boxes* asociados a fin de que el framework realizara el cropping sobre las imágenes antes de ser usadas por la red.
4. Finalmente se realizó un **estudio experimental** en el cual se probaron diversas técnicas para predecir los landmarks. En primer lugar se optó por entrenar las ITLs de la red a partir del conocimiento previo de la red en el dataset AFLW (elegido por compartir características con el nuestro), se vieron que los resultados eran buenos pero los errores de reconstrucción altos. Tras esto se optó por tratar de reducir los errores de reconstrucción a ver si así mejoraba el marcado de landmarks, se entrenaron las ITLs de la red junto con el *encoder* dejando congelados los pesos del *decoder* y luego se repitió el mismo experimento pero entrenando el *decoder* y congelando el *encoder*. Los resultados de estos experimentos no fueron satisfactorios pues el error de reconstrucción apenas bajó y el marcado de landmarks no mejoró. Finalmente se optó por aplicar técnicas de *Data augmentation* sobre el dataset original duplicándolo en tamaño. Realizando traslaciones, rotaciones y occlusiones parciales de forma aleatoria sobre cada imagen. Este último modelo fue entrenado durante 80 épocas en total y obtuvo una mejora considerable de los resultados con respecto a los anteriores y fue el elegido como modelo final. Los resultados por landmark pueden consultarse en la [Tabla 9.5](#).

### 10.2. Trabajos Futuros y comentarios

Ante los resultados obtenidos en esta investigación, posibles vías de trabajo futuras podrían ser:

1. La integración del modelo en un proceso real de detección del landmarks cefalométricos por parte de un equipo de expertos y estudiar su rendimiento y precisión.
2. Realizar un ajuste fino previo de la parte no supervisada de la red para aprender a mejorar la reconstrucción de caras en imágenes de baja calidad, en escalas de grises y en distintas posiciones y ver el impacto de la mejora de la reconstrucción de las imágenes en el marcado de landmarks.
3. Extender el conjunto de datos proporcionado aplicando diversas técnicas de *data augmentation* a cada imagen del dataset, con el fin de mejorar aún más la precisión del marcado con más ejemplos de entrenamiento para el modelo.

Para finalizar, la realización de este trabajo fin de grado ha sido todo un reto. Hemos abordado un problema de la vida real abierto, del cual no hay artículos publicados y apenas técnicas que automaticen dicho proceso. El trabajo me ha permitido combinar todo el conocimiento adquirido en la carrera, sobre todo en el ámbito del aprendizaje automático y la visión por

computador con otras nuevas destrezas propias de un trabajo de iniciación a la investigación y ponerlo en práctica con un caso práctico real, así como aprender a tomar decisiones sobre distintos procesos del problema. También he aprendido que en la vida real no siempre vamos a contar con todos los recursos necesarios para poder experimentar todo lo que se quiera, esto lleva a tener que decidir sobre qué vale la pena y que no explorar. Podemos concluir con este trabajo, que el aprendizaje automático, y más concretamente el *deep learning*, tiene herramientas muy potentes que pueden ser de gran utilidad para tareas relacionadas con el procesamiento de imágenes. Como vimos, hay muchos artículos relacionados con el marcado de landmarks en imágenes por medio de técnicas de *deep learning*. Sin embargo, estas herramientas también pueden tener un gran rendimiento en tareas más especializadas como por ejemplo en el marcado de landmarks *cefalométricos* dentro del ámbito de la Antropología Forense y sin tener que contar con un gran número de ejemplos de entrenamiento, como ha quedado de manifiesto en los resultados obtenidos.



## A. Apéndice A

### A.1. Resultados por imagen durante el entrenamiento del modelo base

Tabla A.1.: Predicciones cross-validation modelo base. Primera partición.

ID	Media NME	Error de Reconstrucción
0	1.535	59.029
1	1.56	49.309
2	4.183	48.867
3	2.612	48.995
4	2.292	37.154
5	2.183	36.851
6	1.387	43.12
7	2.03	57.298
8	2.512	30.655
9	1.325	53.025
10	2.534	45.171
11	2.548	31.648
12	3.932	39.924
13	1.878	38.365
14	2.009	29.392
15	1.918	32.506
16	1.604	33.544
17	1.949	28.871
18	3.066	34.425
19	3.175	28.18
20	1.507	54.29
21	2.197	22.775
22	3.922	27.938
23	1.848	31.185
24	4.623	24.655
25	2.586	25.516

*A. Apéndice A*

Tabla A.2.: Predicciones cross-validation modelo base. Segunda partición.

ID	Media NME	Error de Reconstrucción
26	1.69	38.902
27	4.096	54.938
28	1.993	24.286
29	2.062	32.746
30	2.11	36.961
31	3.894	38.756
32	3.197	32.132
33	3.226	47.616
34	1.85	27.53
35	4.284	41.67
36	4.873	39.631
37	2.788	19.852
38	1.716	43.37
39	3.104	36.613
40	12.051	38.841
41	2.253	25.26
42	2.18	39.435
43	1.876	35.23
44	4.051	37.037
45	2.481	20.661
46	2.233	28.849
47	1.917	21.42
48	2.396	30.455
49	2.205	45.255
50	3.188	37.816
51	2.157	43.994

*A.1. Resultados por imagen durante el entrenamiento del modelo base*

Tabla A.3.: Predicciones cross-validation modelo base. Tercera partición.

ID	Media NME	Error de Reconstrucción
52	6.354	43.496
53	2.591	28.569
54	13.296	41.94
55	1.947	31.216
56	1.523	40.274
57	4.78	42.132
58	2.847	37.619
59	2.129	47.622
60	1.455	41.383
61	1.672	44.416
62	1.83	42.827
63	1.553	52.38
64	3.872	51.004
65	1.325	42.414
66	1.888	37.104
67	2.124	48.9
68	2.161	72.624
69	2.378	69.777
70	2.045	34.079
71	1.993	38.541
72	1.701	24.832
73	2.452	46.039
74	2.728	46.884
75	1.628	30.904
76	2.026	30.559
77	2.354	31.542

*A. Apéndice A*

Tabla A.4.: Predicciones cross-validation modelo base. Cuarta partición.

ID	Media NME	Error de Reconstrucción
78	2.569	55.734
79	1.815	27.689
80	2.074	30.204
81	2.17	41.558
82	5.085	45.978
83	2.748	21.946
84	1.715	41.306
85	3.631	39.967
86	1.859	40.419
87	3.551	54.48
88	2.874	43.632
89	2.032	27.707
90	2.079	28.605
91	1.416	44.906
92	2.336	35.898
93	8.079	50.387
94	1.684	35.117
95	2.287	29.609
96	1.745	42.435
97	2.001	73.033
98	1.591	38.801
99	2.173	30.82
100	1.334	45.025
101	1.557	34.025
102	1.783	23.429
103	1.623	26.619

*A.1. Resultados por imagen durante el entrenamiento del modelo base*

Tabla A.5.: Predicciones cross-validation modelo base. Cuarta partición.

ID	Media NME	Error de Reconstrucción
104	4.771	31.75
105	1.65	44.35
106	2.757	32.826
107	8.494	38.213
108	2.073	32.732
109	1.718	50.615
110	1.378	24.202
111	2.345	39.788
112	1.45	48.653
113	1.657	34.597
114	2.449	29.845
115	2.016	25.377
116	13.343	27.322
117	3.36	28.844
118	31.908	26.396
119	2.898	34.194
120	2.185	25.476
121	3.302	29.49
122	2.316	35.292
123	1.874	44.414
124	2.003	49.988
125	3.95	32.894
126	2.541	62.936
127	2.183	36.35
128	1.723	48.889
129	2.437	22.253
130	1.609	25.704



## B. Apéndice B

### B.1. Resultados por imagen durante el entrenamiento del modelo de entrenamiento del encoder

Tabla B.1.: Tabla con los resultados por imagen obtenidos en validación para la primera partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	1.917	59.224
1.0	1.601	48.894
2.0	4.7	48.865
3.0	3.051	48.772
4.0	2.204	36.999
5.0	2.248	36.486
6.0	1.752	43.42
7.0	2.067	57.024
8.0	2.219	30.838
9.0	1.371	52.661
10.0	2.39	44.856
11.0	3.138	31.588
12.0	4.299	40.125
13.0	2.876	38.606
14.0	2.007	29.139
15.0	1.921	32.303
16.0	1.4	33.311
17.0	1.88	28.922
18.0	3.6	34.225
19.0	3.138	28.305
20.0	1.296	54.197
21.0	2.009	22.738
22.0	4.088	28.035
23.0	1.85	31.188
24.0	4.686	24.579
25.0	2.687	25.556

*B. Apéndice B*

Tabla B.2.: Tabla con los resultados por imagen obtenidos en validación para la segunda partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	<b>1.424</b>	<b>39.145</b>
1.0	<b>4.347</b>	<b>55.074</b>
2.0	<b>1.942</b>	<b>24.493</b>
3.0	<b>2.07</b>	<b>32.785</b>
4.0	<b>2.033</b>	<b>37.225</b>
5.0	<b>3.657</b>	<b>38.844</b>
6.0	<b>2.907</b>	<b>32.204</b>
7.0	<b>3.203</b>	<b>47.254</b>
8.0	<b>1.754</b>	<b>27.563</b>
9.0	<b>4.244</b>	<b>41.532</b>
10.0	<b>4.496</b>	<b>39.766</b>
11.0	<b>2.949</b>	<b>19.764</b>
12.0	<b>2.032</b>	<b>43.229</b>
13.0	<b>2.902</b>	<b>36.356</b>
14.0	<b>12.066</b>	<b>39.045</b>
15.0	<b>2.335</b>	<b>25.173</b>
16.0	<b>2.34</b>	<b>39.39</b>
17.0	<b>1.858</b>	<b>35.206</b>
18.0	<b>3.63</b>	<b>37.226</b>
19.0	<b>2.401</b>	<b>20.641</b>
20.0	<b>2.196</b>	<b>28.644</b>
21.0	<b>2.241</b>	<b>21.356</b>
22.0	<b>2.354</b>	<b>30.385</b>
23.0	<b>2.042</b>	<b>45.022</b>
24.0	<b>2.899</b>	<b>37.786</b>
25.0	<b>2.13</b>	<b>44.0</b>

*B.1. Resultados por imagen durante el entrenamiento del modelo de entrenamiento del encoder*

Tabla B.3.: Tabla con los resultados por imagen obtenidos en validación para la tercera partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	7.06	43.425
1.0	2.554	28.555
2.0	12.688	41.915
3.0	1.827	30.817
4.0	1.548	39.959
5.0	3.624	42.032
6.0	2.775	37.252
7.0	2.015	47.492
8.0	1.56	41.173
9.0	1.629	44.076
10.0	1.739	42.661
11.0	1.439	51.864
12.0	4.549	50.651
13.0	1.353	42.501
14.0	1.941	36.81
15.0	1.922	49.144
16.0	2.302	72.716
17.0	2.137	69.931
18.0	1.691	34.2
19.0	1.851	38.853
20.0	1.746	25.028
21.0	2.672	46.017
22.0	2.9	46.945
23.0	1.523	30.754
24.0	2.097	30.42
25.0	2.316	31.522

*B. Apéndice B*

Tabla B.4.: Tabla con los resultados por imagen obtenidos en validación para la cuarta partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	2.154	56.145
1.0	1.81	27.876
2.0	2.001	30.508
3.0	2.185	41.829
4.0	5.209	46.52
5.0	2.363	21.926
6.0	1.611	41.132
7.0	3.391	40.146
8.0	1.699	40.159
9.0	3.795	54.544
10.0	2.872	43.514
11.0	1.86	27.676
12.0	1.993	28.741
13.0	1.418	45.016
14.0	2.208	36.163
15.0	7.974	49.997
16.0	1.644	35.049
17.0	2.019	29.871
18.0	1.762	42.234
19.0	2.101	73.189
20.0	1.659	38.425
21.0	1.886	30.874
22.0	1.346	44.76
23.0	1.745	33.919
24.0	1.656	23.325
25.0	1.618	26.856

*B.1. Resultados por imagen durante el entrenamiento del modelo de entrenamiento del encoder*

Tabla B.5.: Tabla con los resultados por imagen obtenidos en validación para la quinta partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	5.533	31.555
1.0	1.521	44.151
2.0	2.917	32.84
3.0	8.755	38.414
4.0	2.119	32.667
5.0	1.756	50.634
6.0	1.429	24.228
7.0	2.301	39.335
8.0	1.599	48.683
9.0	1.581	34.454
10.0	2.163	29.64
11.0	2.508	25.183
12.0	13.395	27.301
13.0	3.217	28.529
14.0	31.308	26.59
15.0	3.007	34.393
16.0	2.167	25.252
17.0	3.956	29.492
18.0	2.371	35.544
19.0	1.962	44.177
20.0	1.902	50.071
21.0	3.87	32.492
22.0	2.407	63.048
23.0	2.21	36.392
24.0	1.648	49.032
25.0	2.498	22.041
26.0	1.798	25.626



## C. Apéndice C

### C.1. Resultados por imagen durante el entrenamiento del modelo de entrenamiento del decoder

Tabla C.1.: Tabla con los resultados por imagen obtenidos en validación para la primera partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	1.74	59.029
1.0	1.495	49.309
2.0	4.38	48.867
3.0	3.608	48.995
4.0	2.305	37.154
5.0	2.171	36.851
6.0	1.249	43.12
7.0	2.173	57.298
8.0	2.521	30.655
9.0	1.421	53.025
10.0	2.343	45.171
11.0	2.757	31.648
12.0	4.683	39.924
13.0	3.019	38.365
14.0	1.912	29.392
15.0	2.075	32.506
16.0	1.507	33.544
17.0	1.88	28.871
18.0	2.943	34.425
19.0	3.625	28.18
20.0	1.435	54.29
21.0	2.245	22.775
22.0	3.629	27.938
23.0	1.974	31.185
24.0	4.935	24.655
25.0	2.63	25.516

### C. Apéndice C

Tabla C.2.: Tabla con los resultados por imagen obtenidos en validación para la segunda partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	<b>1.541</b>	38.902
1.0	<b>4.566</b>	54.938
2.0	<b>1.99</b>	24.286
3.0	<b>2.282</b>	32.746
4.0	<b>2.102</b>	36.961
5.0	<b>4.094</b>	38.756
6.0	<b>2.949</b>	32.132
7.0	<b>3.795</b>	47.616
8.0	<b>1.903</b>	27.53
9.0	<b>4.455</b>	41.67
10.0	<b>4.723</b>	39.631
11.0	<b>2.78</b>	19.852
12.0	<b>2.117</b>	43.37
13.0	<b>3.331</b>	36.613
14.0	<b>12.22</b>	38.841
15.0	<b>2.339</b>	25.26
16.0	<b>2.438</b>	39.435
17.0	<b>1.852</b>	35.23
18.0	<b>3.785</b>	37.037
19.0	<b>2.073</b>	20.661
20.0	<b>2.125</b>	28.849
21.0	<b>2.226</b>	21.42
22.0	<b>2.325</b>	30.455
23.0	<b>2.15</b>	45.255
24.0	<b>2.61</b>	37.816
25.0	<b>2.328</b>	43.994

*C.1. Resultados por imagen durante el entrenamiento del modelo de entrenamiento del decoder*

Tabla C.3.: Tabla con los resultados por imagen obtenidos en validación para la tercera partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	6.784	43.496
1.0	2.543	28.569
2.0	13.249	41.94
3.0	1.942	31.216
4.0	1.699	40.274
5.0	3.863	42.132
6.0	2.699	37.619
7.0	2.202	47.622
8.0	1.603	41.383
9.0	1.661	44.416
10.0	1.675	42.827
11.0	1.467	52.38
12.0	4.13	51.004
13.0	1.395	42.414
14.0	2.306	37.104
15.0	1.929	48.9
16.0	2.113	72.624
17.0	2.275	69.777
18.0	1.929	34.079
19.0	1.896	38.541
20.0	1.78	24.832
21.0	2.869	46.039
22.0	3.001	46.884
23.0	1.855	30.904
24.0	2.19	30.559
25.0	2.895	31.542

### C. Apéndice C

Tabla C.4.: Tabla con los resultados por imagen obtenidos en validación para la cuarta partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	2.513	55.734
1.0	1.632	27.689
2.0	1.927	30.204
3.0	2.423	41.558
4.0	4.714	45.978
5.0	2.328	21.946
6.0	1.486	41.306
7.0	2.908	39.967
8.0	1.706	40.419
9.0	3.82	54.48
10.0	2.51	43.632
11.0	2.08	27.707
12.0	2.146	28.605
13.0	1.743	44.906
14.0	2.388	35.898
15.0	7.939	50.387
16.0	1.463	35.117
17.0	2.296	29.609
18.0	1.836	42.435
19.0	2.317	73.033
20.0	1.901	38.801
21.0	1.878	30.82
22.0	1.45	45.025
23.0	1.859	34.025
24.0	1.817	23.429
25.0	1.673	26.619

*C.1. Resultados por imagen durante el entrenamiento del modelo de entrenamiento del decoder*

Tabla C.5.: Tabla con los resultados por imagen obtenidos en validación para la quinta partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	4.835	31.75
1.0	1.596	44.35
2.0	3.057	32.826
3.0	12.16	38.213
4.0	2.441	32.732
5.0	1.883	50.615
6.0	1.49	24.202
7.0	2.171	39.788
8.0	1.549	48.653
9.0	1.4	34.597
10.0	2.588	29.845
11.0	2.336	25.377
12.0	13.375	27.322
13.0	3.405	28.844
14.0	31.204	26.396
15.0	2.717	34.194
16.0	2.035	25.476
17.0	3.081	29.49
18.0	2.329	35.292
19.0	1.848	44.414
20.0	2.068	49.988
21.0	4.124	32.894
22.0	2.249	62.936
23.0	2.052	36.35
24.0	1.491	48.889
25.0	2.483	22.253
26.0	1.734	25.704



## D. Apéndice D

### D.1. Resultados por imagen durante el entrenamiento del modelo con data augmentation

Tabla D.1.: Tabla con los resultados por imagen obtenidos en validación para la primera partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	1.112	59.029
1.0	1.262	49.309
2.0	2.743	48.867
3.0	1.113	48.995
4.0	1.457	37.154
5.0	1.126	36.851
6.0	0.847	43.12
7.0	1.582	57.298
8.0	0.961	30.655
9.0	0.871	53.025
10.0	1.014	45.171
11.0	1.454	31.648
12.0	0.953	39.924
13.0	1.232	38.365
14.0	1.209	29.392
15.0	1.143	32.506
16.0	1.188	33.544
17.0	1.355	28.871
18.0	1.032	34.425
19.0	1.889	28.18
20.0	1.115	54.29
21.0	1.382	22.775
22.0	1.651	27.938
23.0	1.253	31.185
24.0	2.493	24.655
25.0	1.363	25.516

*D. Apéndice D*

Tabla D.2.: Tabla con los resultados por imagen obtenidos en validación para la segunda partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	<b>1.291</b>	38.902
1.0	<b>1.062</b>	54.938
2.0	<b>1.507</b>	24.286
3.0	<b>1.213</b>	32.746
4.0	<b>1.655</b>	36.961
5.0	<b>1.582</b>	38.756
6.0	<b>1.538</b>	32.132
7.0	0.97	47.616
8.0	<b>1.34</b>	27.53
9.0	<b>4.258</b>	41.67
10.0	<b>3.716</b>	39.631
11.0	<b>2.012</b>	19.852
12.0	<b>1.25</b>	43.37
13.0	0.81	36.613
14.0	<b>7.804</b>	38.841
15.0	<b>1.336</b>	25.26
16.0	<b>1.26</b>	39.435
17.0	<b>1.328</b>	35.23
18.0	<b>0.827</b>	37.037
19.0	<b>1.328</b>	20.661
20.0	<b>1.069</b>	28.849
21.0	<b>1.146</b>	21.42
22.0	<b>1.191</b>	30.455
23.0	<b>1.374</b>	45.255
24.0	<b>1.86</b>	37.816
25.0	<b>1.125</b>	43.994

*D.1. Resultados por imagen durante el entrenamiento del modelo con data augmentation*

Tabla D.3.: Tabla con los resultados por imagen obtenidos en validación para la segunda partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	6.702	43.496
1.0	2.91	28.569
2.0	12.538	41.94
3.0	1.799	31.216
4.0	1.642	40.274
5.0	4.357	42.132
6.0	2.693	37.619
7.0	2.297	47.622
8.0	1.657	41.383
9.0	1.71	44.416
10.0	1.649	42.827
11.0	1.487	52.38
12.0	3.994	51.004
13.0	1.194	42.414
14.0	1.972	37.104
15.0	1.995	48.9
16.0	2.042	72.624
17.0	2.496	69.777
18.0	1.89	34.079
19.0	2.157	38.541
20.0	1.601	24.832
21.0	3.828	46.039
22.0	3.316	46.884
23.0	1.822	30.904
24.0	2.2	30.559
25.0	2.363	31.542

*D. Apéndice D*

Tabla D.4.: Tabla con los resultados por imagen obtenidos en validación para la cuarta partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	1.187	55.734
1.0	1.084	27.689
2.0	1.249	30.204
3.0	1.062	41.558
4.0	4.391	45.978
5.0	1.498	21.946
6.0	1.056	41.306
7.0	0.96	39.967
8.0	1.309	40.419
9.0	1.568	54.48
10.0	1.464	43.632
11.0	1.543	27.707
12.0	1.197	28.605
13.0	1.207	44.906
14.0	1.687	35.898
15.0	6.991	50.387
16.0	0.88	35.117
17.0	1.212	29.609
18.0	1.346	42.435
19.0	1.128	73.033
20.0	1.109	38.801
21.0	1.251	30.82
22.0	1.103	45.025
23.0	1.001	34.025
24.0	1.224	23.429
25.0	1.025	26.619

*D.1. Resultados por imagen durante el entrenamiento del modelo con data augmentation*

Tabla D.5.: Tabla con los resultados por imagen obtenidos en validación para la quinta partición de cross validation en la última validación.

ID imagen	Media NME	Error de Reconstrucción
0.0	3.554	31.75
1.0	1.359	44.35
2.0	0.846	32.826
3.0	4.641	38.213
4.0	1.339	32.732
5.0	1.077	50.615
6.0	1.15	24.202
7.0	1.437	39.788
8.0	1.098	48.653
9.0	0.959	34.597
10.0	0.995	29.845
11.0	0.958	25.377
12.0	12.817	27.322
13.0	1.986	28.844
14.0	7.361	26.396
15.0	1.845	34.194
16.0	0.652	25.476
17.0	0.859	29.49
18.0	1.024	35.292
19.0	1.164	44.414
20.0	0.978	49.988
21.0	1.38	32.894
22.0	1.324	62.936
23.0	0.947	36.35
24.0	0.969	48.889
25.0	1.181	22.253
26.0	0.907	25.704



## Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

- [AIA<sup>+</sup>14] Salina Mohd Asi, Nor Hidayah Ismail, Roshahida Ahmad, Effirul Ikhwan Ramlan, and Zainal Arif Abdul Rahman. Automatic craniofacial anthropometry landmarks detection and measurements for the orbital region. *Procedia Computer Science*, 42:372–377, 2014. [Citado en pág. 83]
- [BM13] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013. [Citado en págs. 22 and 27]
- [BW20] Bjorn Browatzki and Christian Wallraven. 3fabrec: Fast few-shot face alignment by reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6110–6120, 2020. [Citado en págs. 49, 87, 94, 95, and 111]
- [DCI20] Sergio Damas, Oscar Cordón, and Oscar Ibáñez. *Handbook on craniofacial superimposition: The MEPROCS project*. Springer Nature, 2020. [Citado en pág. 48]
- [Der17] Arden Dertat. Applied deep learning. = <https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798>, 2017. [Citado en págs. 74, 75, and 76]
- [Fei17] Fei-Fei Li, Justin Johnson, Serena Young , Stanford University. cs231n. = <http://cs231n.stanford.edu/2017/syllabus.html>, 2017. [Citado en págs. 61, 65, 66, 68, 69, 72, 73, and 74]
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>. [Citado en pág. 62]
- [GFCS15] Marek Galvánek, Katarína Furmanová, Igor Chalás, and Jiří Sochor. Automated facial landmark detection, comparison and visualization. In *Proceedings of the 31st spring conference on computer graphics*, pages 7–14, 2015. [Citado en págs. 83 and 84]
- [Gon17] Rafael C. González. *Digital image processing / Rafael C. Gonzalez, Richard E. Woods*. Pearson Education Limited, Harlow, 4th ed., global ed. edition, 2017. [Citado en pág. 7]
- [Gup20] Aaryan Gupta. Evolution of convolutional neural network architectures. = <https://medium.com/the-pen-point/evolution-of-convolutional-neural-network-architectures-6b9odo67e403>, 2020. [Citado en pág. 70]
- [HIWK15] María Isabel Huete, Óscar Ibáñez, Caroline Wilkinson, and Tzipi Kahana. Past, present, and future of craniofacial superimposition: Literature and international surveys. *Legal medicine*, 17 4:267–78, 2015. [Citado en pág. 48]
- [HNATT22] Nguyen Dao Xuan Hai Ho Nguyen Anh Tuan and Nguyen Truong Thinh. The improved faster r-cnn for detecting small facial landmarks on vietnamese human face based on clinical diagnosis. *Journal of Image and Graphics(United Kingdom)*, 12:76–81, 2022. [Citado en pág. 85]
- [HZRS16] 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*, pages 770–778, 2016. [Citado en págs. 73 and 74]
- [J.B12] J.Bruna. Operators commuting with diffeomorphisms. *CMAP Tech. REport*, 2012. [Citado en pág. 22]

## Bibliografía

- [KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. [Citado en pág. 71]
- [KWRB11] Martin Köstinger, Paul Wohlhart, Peter M. Roth, and Horst Bischof. Annotated facial landmarks in the wild: A large-scale, real-world database for facial landmark localization. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 2144–2151, 2011. [Citado en pág. 93]
- [LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. [Citado en págs. 70 and 71]
- [LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. [Citado en pág. 27]
- [Low04] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60, 2004. [Citado en pág. 28]
- [Maloo] Stéphane Mallat. *Une exploration des signaux en ondelettes*. Palaiseau: Les Éditions de l’École Polytechnique, 2000. [Citado en págs. 12 and 14]
- [Mal12] Stéphane Mallat. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65, 10 2012. [Citado en págs. 7, 22, 26, and 30]
- [MMi<sup>+</sup>20] Pablo Mesejo, Ruben Martos, Oscar ibáñez, Jorge Novo, and Marcos Ortega. A survey on artificial intelligence techniques for biomedical image analysis in skeleton-based forensic human identification. *Applied Sciences*, 10:4703, 07 2020. [Citado en pág. 48]
- [NIo2] P Russel Norvig and S Artificial Intelligence. *A modern approach*, volume 90. 2002. [Citado en pág. 47]
- [PJDoMo06] University of Iowa Palle Jorgensen Department of Mathematics. Image decomposition using haar wavelet. = <https://homepage.divms.uiowa.edu/~jorgen/Haar.html>, 2006. [Citado en pág. 15]
- [PLF<sup>+</sup>19] Lucas Faria Porto, Laise Nascimento Correia Lima, Marta Regina Pinheiro Flores, Andrea Valsecchi, Oscar Ibanez, Carlos Eduardo Machado Palhares, and Flavio de Barros Vidal. Automatic cephalometric landmarks detection on frontal faces: An approach based on supervised learning techniques. *Digital Investigation*, 30:108–116, 2019. [Citado en pág. 84]
- [Ras20] Fathy Rashad. Adversarial auto encoder (aae). = <https://medium.com/vitrox-publication/adversarial-auto-encoder-aae-a3fc86f71758>, 2020. [Citado en págs. 77 and 79]
- [Roc19a] Joseph Rocca. Understanding generative adversarial networks (gans). = <https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29>, 2019. [Citado en pág. 75]
- [Roc19b] Joseph Rocca. Understanding variational autoencoders (vaes). = <https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73>, 2019. [Citado en págs. 77 and 78]
- [Ros88] Azriel Rosenfeld. Computer vision: basic principles. *Proceedings of the IEEE*, 76(8):863–868, 1988. [Citado en pág. 61]
- [SLJ<sup>+</sup>15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. [Citado en pág. 72]
- [SSA17] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017. [Citado en págs. 63 and 64]

- [STZP13] Christos Sagonas, Georgios Tzimiropoulos, Stefanos Zafeiriou, and Maja Pantic. 300 faces in-the-wild challenge: The first facial landmark localization challenge. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 397–403, 2013. [Citado en págs. 93 and 94]
- [SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. [Citado en pág. 73]
- [TLF10] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide baseline stereo. *IEEE transactions on pattern analysis and machine intelligence*, 32:815–30, 05 2010. [Citado en pág. 28]
- [WBSSo4] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. [Citado en pág. 104]
- [Wor] Math Works. Continuous wavelet transform and scale-based analysis. [=https://www.mathworks.com/help/wavelet/gs/continuous-wavelet-transform-and-scale-based-analysis.html](https://www.mathworks.com/help/wavelet/gs/continuous-wavelet-transform-and-scale-based-analysis.html). [Citado en pág. 17]