UNIVERSIDAD
DE GRANADA

Facultad de CienciasLocalización de landmarks cefalométricos por medio de técnicas de few-shot learning

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

Tutor:
Pablo Mesejo Santiago
DECSAI

Guillermo Gómez Trenado
DECSAI

Javier Merí de la Maza
Dpto Análisis Matemático

Curso académico 2021-2022

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 2021-2022.

Responsable de tutorización

Pablo Mesejo Santiago
DECSAI

Guillermo Gómez Trenado
DECSAI

Javier Merí de la Maza
Dpto Análisis Matemático

Doble Grado en Ingeniería Informática y Matemáticas

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 2021-2022, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 27 de septiembre de 2022

Fdo: Alejandro Borrego Megías

Dedicatoria (opcional)
Ver archivo `preliminares/dedicatoria.tex`

Índice general

Agradecimientos

Agradecimientos del libro (opcional, ver archivo `preliminares/agradecimiento.tex`).

Summary

An english summary of the project (around 800 and 1500 words are recommended).
File: `preliminares/summary.tex`

Introducción

De acuerdo con la comisión de grado, el TFG debe incluir una introducción en la que se describan claramente los objetivos previstos inicialmente en la propuesta de TFG, indicando si han sido o no alcanzados, los antecedentes importantes para el desarrollo, los resultados obtenidos, en su caso y las principales fuentes consultadas.

Ver archivo `preliminares/introduccion.tex`

Parte I.

Análisis de Redes Convolucionales

Si el trabajo se divide en diferentes partes es posible incluir al inicio de cada una de ellas un breve resumen que indique el contenido de la misma. Esto es opcional.

1. Introducción

Actualmente, las **Redes Neuronales Convolucionales** (CNN[1]) son una de las herramientas más usadas de la Inteligencia Artificial para tareas de Aprendizaje Automático (AA), de hecho son el principal objeto de estudio del **Deep Learning**, una rama del AA en la que hoy en día se está invirtiendo mucho esfuerzo en invesitgar y de la que anualmente se publican muchos artículos que nos enseñan la gran potencia de las CNN para diveras tareas.

Destaca especialmente el excelente desempeño que tienen las CNN 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 estas CNN, para conocerlas mejor desde un punto de vista más teórico y **poder demostrar una de sus principales propiedades**: la invarianza por traslaciones.

En primer lugar vamos a definir 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 esto 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. Esta propiedad es fundamental y sabemos que las CNN la verifican.

Figura 1.1.: Las tres estatuas deben identificarse como iguales, aunque se ecuentren desplazadas.

Otra propiedad importante que sería la **invarianza frente a pequeñas deformaciones** (difeomorfismos), 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.

Realizaremos nuestro estudio sobre las funciones $L^2(\mathbb{R}^d)$, que son Lipschitz-continuas por la acción de difeomorfismos y que mantienen información de alta frecuencia para diferenciar entre distintos tipos de señales. Así, el objetivo será encontrar un operador que verifique una serie de propiedades que veremos en secciones futuras así como la invarianza por traslaciones que presentaremos como una posible modelización matemática de una CNN.

[1] Convolutional Neural Network

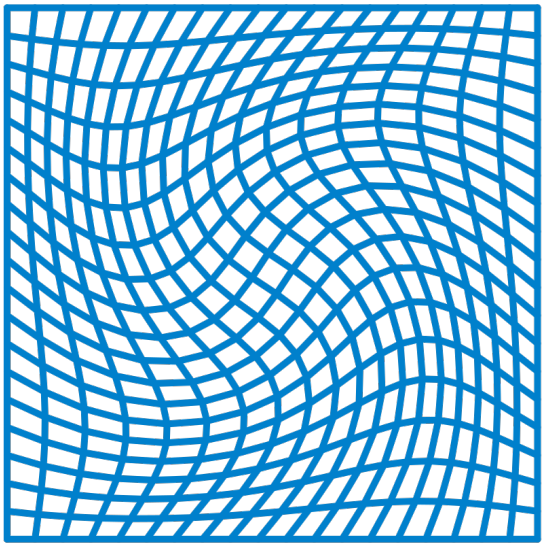


Figura 1.2.: Acción de un difeomorfismo en una rejilla.

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

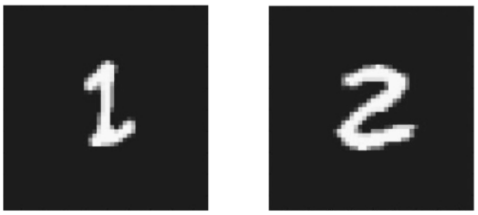


Figura 1.4.: Deformación excesiva que permite confundir el 1 con el 2 cuando se le aplica el difeomorfismo. Por eso nos centramos en *"pequeñas"* deformaciones, para no alterar la identidad del objeto en la imagen.

La invarianza por traslaciones, 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:

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 Φ de $L^2(\mathbb{R}^d)$ en un espacio de Hilbert $\mathcal{H}$ 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 en este caso.

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

- Denotamos por $\|f\|$ a la norma de f en $L^2(\mathbb{R}^d)$.
- Denotamos por $\|f\|_{\mathcal{H}}$ a la norma de f en el espacio de Hilbert $\mathcal{H}$.

Definición 1.0.2. Decimos que Φ es no-expansiva si:

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

Por otro lado:

Definición 1.0.3. Una función diferenciable $f : X \to \Omega$ donde X y Ω son variedades, es un *"Difeomorfismo"* si f es una biyección y su inversa $f^{-1} : \Omega \to X$ 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 τ es el campo de desplazamiento.

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

Por otro lado, la condición de Lipschitz es la siguiente:

Definición 1.0.5. Sea $f : M \to N$ 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:

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

En nuestro caso, la distancia que utilizaremos será la inducida por la norma del espacio de Hilbert $\mathcal{H}$ de llegada, pero necesitamos definir de alguna manera una distancia d_M entre los difeomorfismo $\mathbb{1}$ y $\mathbb{1} - \tau$ para escribir correctamente la condición de Lipschitz. Además, 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 $\|\Phi(f) - \Phi(L_c f)\|$ 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 Ω de $\mathbb{R}^d$ como

$$d_\Omega(\mathbb{1}, \mathbb{1} - \tau) = \sup_{x \in \Omega} |\tau(x)| + \sup_{x \in \Omega} |\nabla \tau(x)| + \sup_{x \in \Omega} |H\tau(x)| \tag{1.1}$$

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

En lo que sigue vamos a denotar:

- $||\tau||_\infty := \sup_{x\in\mathbb{R}^d} |\tau(x)|$
- $||\nabla\tau||_\infty := \sup_{x\in\mathbb{R}^d} |\nabla\tau(x)|$
- $||H\tau||_\infty := \sup_{x\in\mathbb{R}^d} |H\tau(x)|$ donde $|H\tau(x)|$ es la norma del tensor Hessiano.

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 Φ 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 Ω y para todo $\tau \in C^2(\mathbb{R}^d)$ se cumple:

$$||\Phi(f) - \Phi(L_\tau f)||_{\mathcal{H}} \leq C||f||(\sup_{x\in\mathbb{R}^d} |\nabla\tau(x)| + \sup_{x\in\mathbb{R}^d} |H\tau(x)|) \tag{1.2}$$

con $||\nabla\tau||_\infty + ||H\tau||_\infty < 1$ para asegurarnos de que la deformación sea invertible [TY05].

Debido a que Φ es invariante a traslaciones, la cota superior de Lipschitz no depende de la amplitud máxima de traslación $\sup_{x\in\mathbb{R}^d} |\tau(x)|$ de la métrica del difeomorfismo (1.1). Por otro lado la continuidad Lipschitz de (1.2) implica que Φ es invariante por traslaciones globales, pero es mucho más fuerte. Φ 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, en las futuras secciones veremos como para solucionar el problema se optará por utilizar **transformadas de ondeletas**, aunque esto abre nuevos problemas 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** obtener coeficientes invariantes. Este nuevo operador consistirá en una **cascada de convoluciones** de operadores no lineales y no conmuntativos de manera que cada uno de ellos calcula el módulo de la transfomada de odeletas, 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 en primer lugar definir un operador que denominaremos **propagador de dispersión** (PD) que será el que aplicaremos de forma recursiva en la cascada de convoluciones que modeliza una CNN, para ello 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.

Tras esto 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 segunda alternativa obtendremos un operador que es **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 anterior sobre un *"camino"* de frecuencias y rotaciones el que definirá la modelización matemática de una Red Neuronal Convolucional. Todo este capítulo está basado en la investigación de Stéphane 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 bajo la curva finita) como la integral de senos y cosenos multiplicados por una función que determina los pesos en cada instante. Formalmente tiene la siguiente expresión:

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

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"*[1] ya que al calcular la integral, la función resultante sólo depende de ω (la frecuencia), y posteriormente pasar de

[1] También llamado *"Dominio de Frecuencia"*

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 Fourirer para evitar fases complejas en el análisis, de esta forma el operador que vamos a probar en primer lugar es:

Definición 2.1.1. $\Phi(f) = |\widehat{f}|$ 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**. Veamos que sí cumple esta propiedad.

Lema 2.1.2. *El operador $\Phi(f) = |\widehat{f}|$ es invariante por traslaciones.*

Demostración. Para ello tenemos que ver que si definimos 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 tendría que:

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

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

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

Observación 2.1.3. El módulo de la Transformada de Fourier no es estable frente a pequeñas deformaciones y no es *"Lipschitz-continuo"* en el sentido de la Teorema 1.0.7.

Si consideramos la función $\tau(x) := \epsilon x$ con $0 < \epsilon << 1$. 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 transfomada 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 $0 < \epsilon << 1$ sería:

$$\| \, |\widehat{f}| - |\widehat{L_\tau f}| \, \| \leq c\|f\|(\|\nabla\tau\|_\infty + \|H\tau\|_\infty) \leq c\|f\|\epsilon \tag{2.1}$$

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

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

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

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,

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

y

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

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

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

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

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:

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

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 ξ elevado se cumpla que:

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

Y como ξ 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:

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

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 ϵ.

La demostración 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 al caso de una función de soporte compacto.

Para conseguir esto, reemplazaremos las ondas sinusoidales de la transformada de Fourier

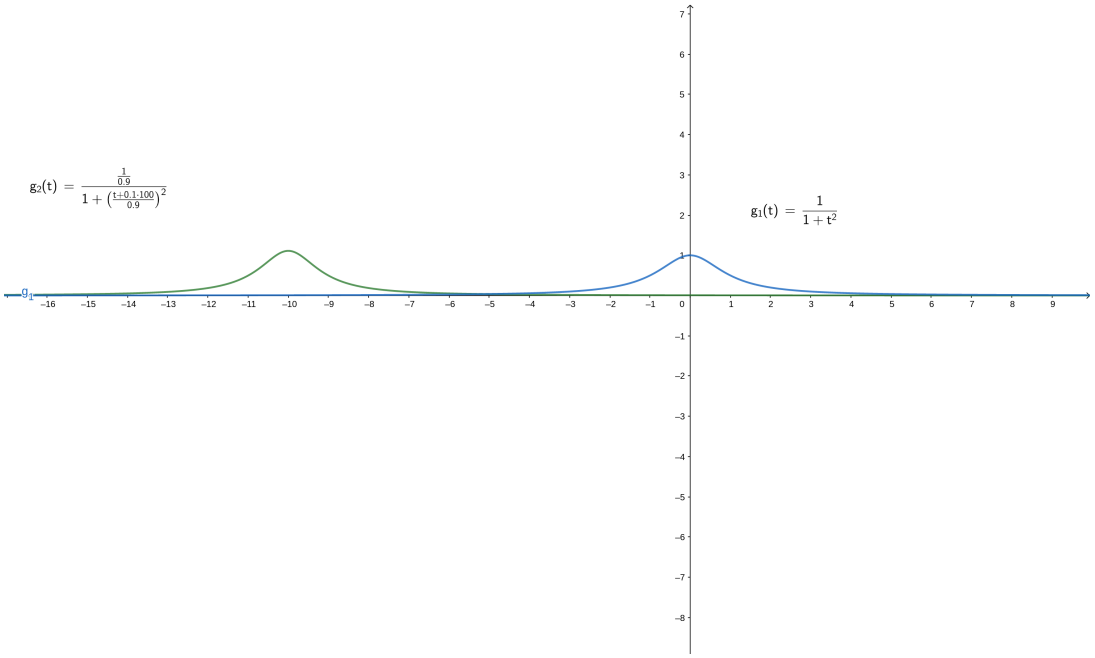


Figura 2.1.: Como podemos ver en la imagen, para los valores $\epsilon = 0.1$, $\xi = 100$ y $M = 5$ ambas funciones tienen soporte casi disjunto de manera que la diferencia entre ellas en el intervalo $[-5, 5]$ coincide prácticamente con $g_1(t)$.

por funciones localizadas con un soporte mayor en altas frecuencias que nos permitan evitar estas complicaciones, que tendrán un mejor rendimiento en nuestro propósito. Estas funciones se denominan **ondeletas**.

2.1.2. Alternativa: Las ondeletas

Las ondeletas [Maloo] son pequeñas ondas estables bajo la acción de deformaciones, al contrario que 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. Se construye a partir de la siguiente función:

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

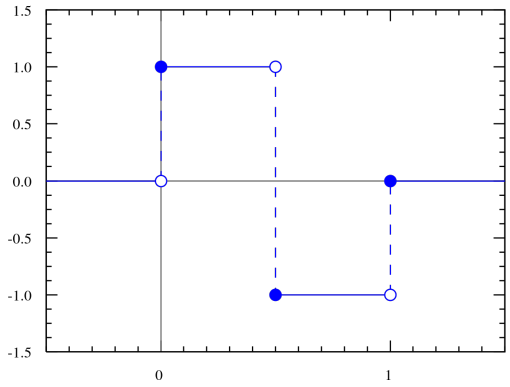


Figura 2.2.: Representación gráfica de la ondeleta de Haar.

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 que nos permite 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 serían:

- 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 dónde tiene soporte la ondeleta correspondiente soporte 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 [2], 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 $L^2(\mathbb{R}^2)$:

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

El soporte de la ondeleta $\psi^k_{j,n}(x)$ 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.

Del mismo modo que en una dimensión, los coeficientes de ondeletas $\langle f, \psi^k_{j,n} \rangle$ 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.

[2] ver por ejemplo sección 1.1 de [Maloo]

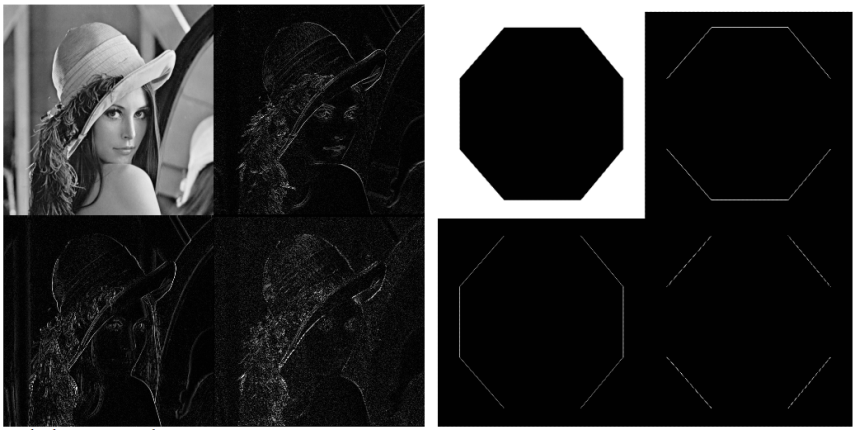


Figura 2.4.: Ejemplos de aplicar la base de Haar a dos imágenes.Los filtros resaltan los bordes en tres direcciones, horizontal (derecha) vertical (abajo) y en diagonal (abajo derecha). Imagen extraida de [PJDoMo6].

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:

Definición 2.1.4. 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:

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

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 centrada en una bola de baja frecuencia en $x = 0$, cuyo radio es del orden de π.

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 centrada en una bola de mismo radio pero centrada en $\omega = \eta$ que tras el escalado y rotación:

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

donde $\lambda = 2^j r \in 2^{\mathbb{Z}} \times G$.
Por lo tanto $\widehat{\psi}_\lambda(\omega)$ recubre una bola centrada en $\lambda^{-1}\eta$ con radio proporcional a $|\lambda| = 2^j$.

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.

Se trata de una representacion redundante que calcula convoluciones para todo $x \in \mathbb{R}^d$ sin realizar sub-muestreo:

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

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, el cual dice:

Teorema 2.1.5. *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 con $\{-\mathbb{1}, \mathbb{1}\}$, 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.

Figura 2.5.: Como podemos ver, en el caso de la izquierda la ondleta (en color azul) 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 mayo, lo que le impedirá detectar con tanta precusion los cambios que se produzcan en la señal. Imagen extraida de [Wor].

- 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.

Con la transformada de Littlewood-Paley ocurre lo mismo, a una cierta escala 2^J (con $j \in \mathbb{Z}$ fijo), sólo retiene las ondeletas de frecuencias $2^j > 2^{-J}$. Así, podemos obtener una estimación de lo dilatadas que deben estar las ondeletas como mínimo para no generar coeficientes no nulos. De esta forma, las bajas frecuencias que no son cubiertas por estas ondeletas vienen dadas por un promedio en el dominio proporcional a 2^J:

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

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 odeletas 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:

$$||W_J f||^2 = ||A_J f||^2 + \sum_{\lambda \in \Lambda_j} ||W[\lambda]f||^2. \tag{2.5}$$

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 $W_J f = \{A_J f, (W[\lambda]f)_{-\lambda,\lambda\in\Lambda_J}\}$ y $W_\infty f = \{W[\lambda]f\}_{-\lambda,\lambda\in\Lambda_\infty}$.

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

Proposición 2.1.6. *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:*

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

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

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

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

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

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

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:

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

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

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

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

con lo que queda demostrado que (2.6) y (2.7) son equivalentes. Teniendo en cuenta que $\widehat{W\left[2^j r\right] f}(\omega) = \widehat{f}(\omega)\widehat{\psi}_{s^j r}(\omega)$, multiplicando (2.7) por $|\widehat{f}(\omega)|^2$ obtenemos:

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

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

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

Si la aplicamos (2.4) se obtiene:

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

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 ω. De no ser así podríamos contruir una función f no nula cuya transformada de fourier $\widehat{f}$ tuviera soporte en el dominio de ω dónde (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 ψ por $\sqrt{2}$ se obtiene la condición (2.6) con $\beta = \frac{1}{2}$. □

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 $|\widehat{\phi}(r\omega)| = |\widehat{\phi}(\omega)| \ \ \forall r \in G$.
- $\widehat{\phi}(\omega)$ es real y simétrica, por lo que ϕ también lo será y $\phi(rx) = \phi(x) \ \ \forall r \in G$.
- Suponemos que ϕ y ψ son dos veces diferenciables y su decrecimineto así como el de sus derivadas de primer y segundo orden es $O((1+|x|)^{-d-2})$.

Un cambio de variable en la integral de la transformada de ondeleta nos muestra que si f se escala y rota, $2^l g \circ f = f(2^l g x)$ 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 ϕ es invariante a traslaciones en G, podemos comprobar que A_J conmuta con las rotaciones de G: $A_J(g \circ f) = g \circ A_J f \ \ \forall g \in G$.

2.2. El operador de dispersión sobre un camino ordenado

Antes de comenzar la sección vamos a aclarar la notación que usaremos de ahora en adelante:

- 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 $\mathcal{R}[\Omega] = \{\mathcal{R}[p]\}_{p \in \Omega}$.

La transformada de Littlewood-Paley definida anteriormente es Lipschitz-conitnua 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 conmuta 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 $U[\lambda]f$ conmuta 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 conmutatividad del operador $U[\lambda]$ se tiene que

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

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. □

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:

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

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 Teorema 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 $M[\lambda]$ conmute con estos 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 [J.B12], 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:

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

Dado que se tratan de dos funciones positivas las que se restan en el integrnado, y para llegar a la conclusión que sugiere el autor [Mal12] se ha supuesto que $|M[\lambda]h(x)| \geq |h(x)| \quad \forall x \in \mathbb{R}^d$.

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.

Por ejemplo, si

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

dónde $\xi_1 > 0$ 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 $\widehat{\psi_{\lambda}}$.

A continuación introducimos el PD que extiende estas descomposiciones.

Definición 2.2.2. Una secuencia ordenada $p = (\lambda_1, \lambda_2, ..., \lambda_m)$ 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 PD es un producto de operadores de la forma $U[\lambda]f = M[\lambda]W[\lambda]f = |f * \psi_{\lambda}| = \left|\int_{\mathbb{R}^d} f(u)\psi_{\lambda}(x-u)du\right|$ para $f \in L^2(\mathbb{R}^d)$ no conmuntativos por un camino ordenado:

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

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:

$$\big||f * \psi_{\lambda_1}| * \psi_{\lambda_2}|...| * \psi_{\lambda_m}\big|$$

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

2.2.3. Propiedades de un camino de frecuencias.

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 todo lo que sabemos sobre caminos, podemos probar la siguiente propiedad:

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

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

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

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

□

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 si f es compleja, tras la iteración $U[\lambda_1]f = |W[\lambda_1]f|$ sería una función real, luego para las siguientes transfomadas de ondeletas solo haría falta calcularlas para $\lambda_k \in \Lambda_\infty$. Por lo tanto para 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 haran sobre PD aplicados a funciones reales.

2.2.4. Construcción del operador de dispersión.

En este momento ya disponemos de un operador $U[\lambda]f$ 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 defnición 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 $\phi_{2^J}(x) = 2^{-dJ}\phi(2^{-J}x)$.

Definición 2.2.8. Sea $J \in \mathbb{Z}$ y $\mathcal{P}_J$ el conjunto de caminos finitos $p = (\lambda_1, \lambda_2, ..., \lambda_m)$ 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 ϕ_{2^J} localiza el propagador de dispersión en dominios proporcionales a 2^J.

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

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 inculir 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 como mencionamos antes, 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 1.14(Teorema 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 ∞:

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

2.3. Propagador de dispersión y conservación de la Norma

2.3.1. Proceso de dispersión del propagador.

Hasta ahora hemos probado que el propagador S_J es no-expansivo y que preserva la norma de $L^2(\mathbb{R}^d)$. A partir de ahora denotamos por $S_J[\Omega] \coloneqq \{S_J[p]\}_{p\in\Omega}$ y $U[\Omega] \coloneqq \{U[p]\}_{p\in\Omega}$ a la familia de operadores indexados por el conjunto de caminos $\Omega \subset \mathcal{P}_\infty$.

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 Teorema 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 Λ_J^m el conjunto de caminos de longitud m con $\Lambda_J^0 = \emptyset$, entonces:

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

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 ∞, tal y cómo se puede ver en la imagen Figura 2.6.

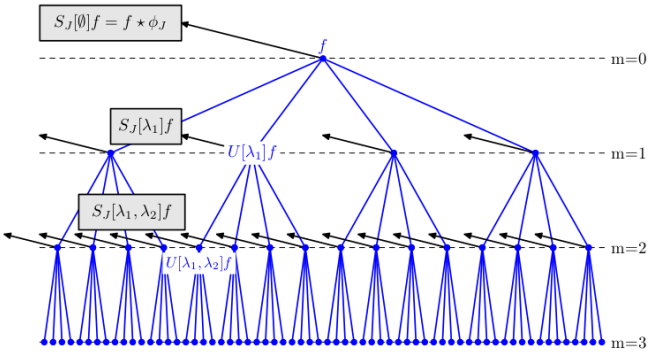


Figura 2.6.: Un PD U_J aplicado a un punto de una señal $f(x)$ calcula $U[\lambda_1]f(x) = |f(x) * \psi_{\lambda_1}|$ y como salida a la capa $m = 0$ se promedian los coeficientes que han dado 0 (por tener $2^j < 2^{-J}$) obteniendo como salida $S_J[\emptyset]f(x) = f(x) * \phi_{2^J}$ (como se puede 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 extraida 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 transfomada 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 ϕ_{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.

2.3.3. Relación con herramientas clásicas de visión por computador

Por otro lado, la modelización con los algoritmos clásicos de visión por computador como **SIFT** [Low04] 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 $S_J[2^j r]f = |f * \psi_{2^j r}| * \phi_{2^J}(x)$, dónde $\psi_{2^j r}$ 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 ϕ_{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 Teorema 2.1.6 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:

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

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 Ω, las normas de $S_J[\Omega]f$ y $U[\Omega]f$ son:

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

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 ∞ 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:

$$\sum_{m=0}^{\infty} ||U[\Lambda_J^0]f - U[\Lambda_J^0]h||^2 = \sum_{m=0}^{\infty} ||U[\emptyset]f - U[\emptyset]h||^2 = ||f - h||^2$$

Por otro lado, 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 transfomada de dispersión de ventana es no expansiva. □

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 alncanza la frecuencia mínima 2^J y es atrapada por el filtro paso bajo ϕ_{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 $\forall f \in L^2(\mathbb{R}^d)$:*

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

Demostración.

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

Dónde se ha usado el cambio de variabel $u' = x - u$ con $Jacobiano = 1$. □

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 $|\widehat{\rho}(\omega)| \leq |\widehat{\phi}(2\omega)|$ y $\widehat{\rho}(0) = 1$, tal que la función:

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

satisface:

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

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:

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

La demostración de lema se encuentra en el apéndice A de [Mal12].

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*

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

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

La clave de esto reside en el Teorema 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 $\lim_{m\to\infty} ||U[\Lambda_J^m]f|| = 0$. 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 $\lim_{n\to-\infty} ||f - f_n|| = 0$, dónde $f_n = f * \phi_{2^n}$ y $\phi_{2^n} = 2^{-nd}\phi(2^{-n}x)$. Se demuestra por tanto que $\lim_{m\to\infty} ||U[\Lambda_J^m]f_n|| = 0$ viendo que $||f_n||_w < \infty$. De hecho,

$$\begin{aligned} ||W[2^j r]f_n||^2 &= \int |\widehat{f}(\omega)|^2 |\widehat{\phi}(2^n\omega)|^2 |\widehat{\psi}(2^{-j}r^{-1}\omega)|^2 d\omega \\ &\leq C2^{-2n-2j} \int |\widehat{f}(\omega)|^2 d\omega, \end{aligned}$$

porque ψ hay un momento en que desaparece entonces $|\widehat{\psi}(\omega)| = O(|\omega|)$, y las derivadas de ϕ 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, $||U[\Lambda_J^m]f - U[\Lambda_J^m]f_n|| \leq ||f - f_n||$, por lo que

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

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

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

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

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

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

Primero probamos que

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

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

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

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

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

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 $m \to \infty$

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

Llegados a este punto se puede apreciar claramente que

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

Y el recíproco también es cierto, luego ambas expresiones son equivalentes.

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

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

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:

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

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

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

$$\Updownarrow$$

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

$$\Updownarrow$$

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

De manera que se puede apreciar claramente que

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

Con lo que queda demostrado el teorema □

2.3.6. Conclusiones extraidas 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 conlleva a que en problemas de clasificación, por ejemplo, el de 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 $\psi(x) = e^{i\eta x}\Theta(x)$ y de hecho $\widehat{\psi} = \widehat{\Theta}(\omega - \eta)$, donde $\widehat{\Theta}$ y $\widehat{\rho}$ tienen su energía concentrada en los mismos dominos 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 invaianza 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 $J \to \infty$. Esto define una distancia límite que como veremos a continuación es invariante por traslaciones.

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

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

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 unívocamente 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:

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

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

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

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 Teorema 2.1.6, para toda función **real** $f \in L^2(\mathbb{R}^d)$ y todo $q \in \mathbb{Z}$ se verifica:*

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

Demostración. En primer lugar vamos a ver que de Teorema 2.1.6 se deduce la siguiente expresión:

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

Para ello, de la expresión

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

se tiene de la misma forma que vimos en la demostración de la Teorema 2.1.6 que:

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

Y partiendo el sumatorio obtenemos que:

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

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

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

Integramos en ω,

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

Ahora estamos en condiciones de aplicar el Teorema 2.1.5, y nos quedaría que la expresión anterior equivale a:

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

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

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

□

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:

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

Y esto equivale a

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

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

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

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 Teorema 2.3.1, usando la desigualdad anterior podemos escribir

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

y en particular

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

que demuestra (3.2). □

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 tampoco:

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

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 $\lim_{J\to\infty} ||S_J[\mathcal{P}_J]f|| = ||f||$. El siguiente teorema demuestra que el límite es invariante por traslaciones, pero para la demostración del teorema necesitaremos un resultado auxiliar:

Lema 3.2.1. *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, al igual que en otras cotas superiores para normas, vamos a necesitar el Lema de Schur [QV18]. De esta manera, el Lema de Schur nos recuerda que para cualquier operador $Kf(x) = \int f(u)k(x,u)du$ se tiene

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

y además

$$\int |k(x,u)|du \leq C \implies ||K|| \leq C.$$

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

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

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

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

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

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

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

Si ahora calculamos el módulo obtenemos que:

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

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

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

por otro lado, vamos a comprobar que

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

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

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

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:

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

De esta forma, realizando un cambio de variable tendríamos:

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

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

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

Si 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

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

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

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

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 a traslaciones.

Teorema 3.2.2. *Para ondeletas de dispersión admisibles se tiene que*

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

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 $S_J[\mathcal{P}_J]f = A_J U[\mathcal{P}_J]f$,

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

Si ahora aplicamos el Teorema 3.2.1 con $\tau = c$, se tiene que $||\tau||_\infty = |c|$ y además

$$||L_c A_J - A_J|| \leq C2^{-J}|c|.$$

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

$$\begin{aligned}||S_J[\mathcal{P}_J]L_c f - S_J[\mathcal{P}_J]f|| &\leq ||L_c A_J - A_J||||U[\mathcal{P}_J]f|| \\ &\leq C2^{-J}|c|||U[\mathcal{P}_J]f||\end{aligned}$$

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

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

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 de antes 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 ambo lados cuando $J \to \infty$ tenemos que

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

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

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, se puede verificar 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 $n \to \infty$ se prueba que $\lim_{J\to\infty} ||S_J[\mathcal{P}_J]f - S_J[\mathcal{P}_J]L_c f|| = 0$ con lo que acaba la demostración. □

4. Conclusiones y Trabajos futuros

En el primer capítulo dimos una introducción a las **CNN** y 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 transfomada de Fourier como candidato por no cumplir esta segunda propiedad.

Una vez rechazado el operador anterior vimos que la alternativa más prometedora era la de usar una transformada de Ondeletas, en concreto la de **Littlewood-Paley**:

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

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:

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

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

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

Con este operador se consigue definir el que será la modelización de una CNN, y que además tiene unas propiedades deseables como la preservación de la norma o que es un operador no expansivo.

Finalmente, con la modelización propuesta (el operador de ventana) se comprueba que no es expansivo en conjuntos de caminos a medida que decrece el umbral y con esta propiedad se consigue demostrar la invarianza por traslaciones del operador.

Llegados a este punto podemos afirmar haber conseguido todos los objetivos que nos habíamos marcado. 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. Es la prmiera vez que me enfrento a un trabajo de iniciación a la investigación, y por lo tanto en muchas situaciones me he visto envuelto de conceptos que no comprendía del todo y de los cuales no había mucha

información viéndome en la necesidad clasificar y seleccionar lo mejor posible la información siempre pensando en los objetivos marcados.

En primer lugar tuve que recordar conceptos básicos del análisis de Fourier como son el cálculo de series de Fourirer y transformada de Fourier así como sus interpretaciones y aplicaciones. Por otro lado tuve que recordar conceptos de análisis como son la Lipschitz-continuidad y con esto pude entender y explicar la demostración de que el módulo de la transformada de Fourier no es lipschitz-continuo bajo la acción de difeomorfismos. También he 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, un mundo desconocido para mi y que es esencial en áreas de conocimiento como el procesamiento de señales y la visión por computador, el cual me ha permitido relacionar directamente los dos grados que he cursado (Informática y Matemáticas). Tuve que aprender y entender el concepto de ondeleta y transformada de ondeletas, y tuve que realizar una investigación sobre el tratamiento de señales (especialmente de imágenes) por medio de estas heramientas.

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 matemática 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

Las **ciencias forenses** son aquellas que aplican el método científico a hechos presuntamente delictivos con la finalidad de aportar pruebas a efectos judiciales. Este campo es interdisciplinar incluye principalmente a la Criminalística[1] y la Medicina Forense[2].

Así pues, este trabajo ubica en el ámbito de la **antropología forense**, que es una rama de la Medicina Forense que se encarga de determinar la edad, raza, sexo o estatura, entre otras, a partir de restos óseos en problemas de reconstrucción facial, identificación de víctimas en desastres en masa o en identificación facial.

5.0.1. Descripción del problema

La **Superposición Craneofacial** es una técnica de identificación forense mediante la cual se comparan imágenes de la persona difunta[3] con una o varias imágenes de un cráneo candidato. La técnica empleada es la superposición de ambas imágenes y se estima si son o no la misma persona de acuerdo a correspondencias morfológicas o marcando puntos de referencia. Los *landmarks* o puntos de referencia, pueden situarse en el cráneo[4] encontrado o en el rostro[5]. Entre los dos tipos de *landmarks* anteriores existe una correlación, en caso de pertenecer a la misma persona, que el antropólogo forense trata de descubrir.
Esta tarea no es sencilla debido al **tejido blando facial** que separa el punto craneométrico de su homólogo cefalométrico y que lo desplaza. El desplazamiento ocasionado por el tejido blando facial no es constante ni se produce siempre en la misma dirección, lo cual junto con otros factores como la grasa o la calidad de la imagen complica esta tarea de superponer las dos imágenes (de cráneo y cara) con fidelidad.

Tradicionalmente, el proceso era esencialmente manual y complicado de replicar, y pese a los avances actuales que se están llevando a cabo para automatizar esta tarea [HIWK15], la identificación de *landmarks* sigue realizándose a mano normalmente.

En este contexto, el presente trabajo se centrará en esta etapa del marcado de *landmarks*, en concreto de **landmarks cefalométricos** (en las imágenes ante-mortem). El objetivo será comparar dos frameworks que utilizan técnicas de **Deep Learning** para la detección y marcado de **landmarks cefalométricos**

5.0.2. Motivación

5.0.3. Objetivos

[1] Disciplina encargada del descubrimiento y verificación científica de presuntos hechos delictivos y quienes los cometen.
[2] Disciplina encargada de determinar el origen de las lesiones, las caisas de muerte o la identificación de seres humanos vivos o muertos.
[3] A estas imágenes se le denominan imágenes ante-mortem
[4] En este caso reciben el nombre de puntos craneométricos
[5] En este caso se denominan puntos cefalométricos

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 más recientemente el conocido **Aprendizaje Automático** (AA).

Los métodos que empleamos en este trabajo pretenecen 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 permite hacernos una idea de lo que se pretende conseguir, que es 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 un poco 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 entrenan. 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 Refuezo que están siendo muy usadas, pero no vamos a profindizar en ellas pues no son necesarias para el trabajo que nos ocupa.

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:

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

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 defnición 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:

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

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

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

y de etiquetas

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

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. 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 principal algoritmo que 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:

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

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

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

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):

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

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 η es un factor conocido como **learning rate**(lr) que mide el *"tamaño"* del paso que en cada iteración damos en búsqueda del mínimo.

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**, pues si este es demasiado pequeño puede ocasionar una lenta convergencia al mínimo, y por lo tanto que se realice un gran número de iteraciones, y en cambio 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).

6.1.1.3. Gradiente Descendente Estocástico

El algoritmo descrito anteriormente tiene el problema de ser costoso computacionalmete, 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"*.

6.1.1.4. 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**, y en ellos se pretende agrupar los datos en dos posibles clases que suelen codificarse 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 pontos más cercanos a él de cada conjunto.
- **Redes Neuronales**, que explicaremos en detalle en futuras secciones y destacando el *perceptrón multicapa* (MLP).

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 los landmarks cefalométricos para una cierta imagen de entrada, lo que nos llevaría a un problema típico de aprendizaje

supervisado en el que pretendemos a partir de la imagen de entrada conocer la función que nos proporciona la salida correcta (la imagen con los landmarks marcados correctametne).

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.

6.2. 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].

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 las **redes convolucionales profundas** que explicaremos en detalle en la siguiente sección. Estas nuevas herramientas han permitido crear programas que obtienen un gran rendimiento en el tratamiento de imágenes.

6.3. 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 odernador, 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 terminos de otras más simples. De esta manera se pueden construir conceptos difíciles a partir de otros más sencillos. Este grafo puede ser tan profundo como se necesite, por ello se le conoce como Deep Learning.

6.3.1. 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 $f'(x; W) = \widehat{y}$ (dónde $\widehat{y}$ 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 $\widehat{y}$. 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.

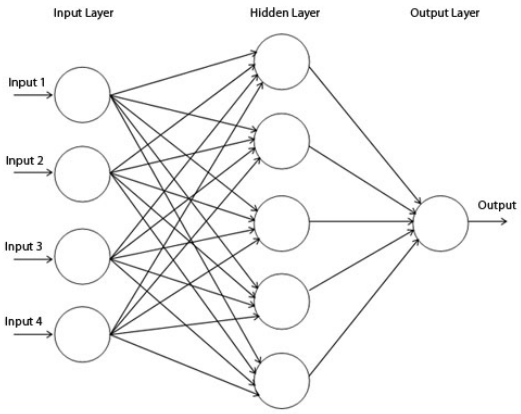


Figura 6.1.: Red neuronal con una capa oculta. Formalmente podría describirse como $f'(x) = (f_3(f_2(f_1(x))))$, dónde f_1 hace referencia a la capa de entrada a la red, f_2 a la capa oculta y f_3 a la capa de salida.

La **profundidad** de la red vendría dada por la cantidad de **capas ocultas** que esta tiene. En los algoritmos de Deep Learning veremos un gran número de capas ocultas, de ahí el nombre.

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 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 Sigmoide**. Transforma los valores de entrada a un valor entre 0 y 1.

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

- **Función Tangente Hiperbólica**. Es similar a la función sigmoide, 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, ya que en aquellas cuya salida tras la combinación lineal con los pesos de la capa correspondiente sea o no serán activadas.

  $$ReLU(x) = \max(0, x)$$

  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 **back-propagation**.

- **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.

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

  Dónde a es un valor muy próximo a 0.

- **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 multiclase, 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 multiclase.

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

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 γ 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.

6.3.2. Back Propagation

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 computacionalmete. En lugar de esto se aplica la técnica de **Back-propagation**.

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 6.2.

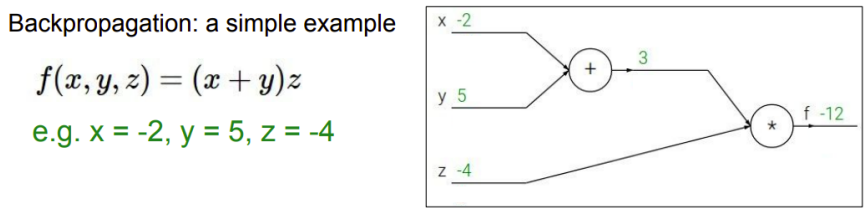


Figura 6.2.: Ejemplo de Grafo computacional junto con la salida para una entrada concreta $x = -2$,$y = 5$, $z = -4$. La imagen ha sido extraida del curso [Fei17].

La idea del algoritmo de Back-propagation 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 quisiéramos 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 6.2.
Como vemos en Figura 6.3, 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**.

6.4. Redes Neuronales Convolucionales

Las **Redes Neuronales Convolucionales** (CNN) 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 una matriz 2D.

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 del volumen de entrada. A continuación vamos a ver qué operación realizaremos con este filtro sobre el volumen de la imagen que ocupa para alimentar cada neurona de la siguiente capa.

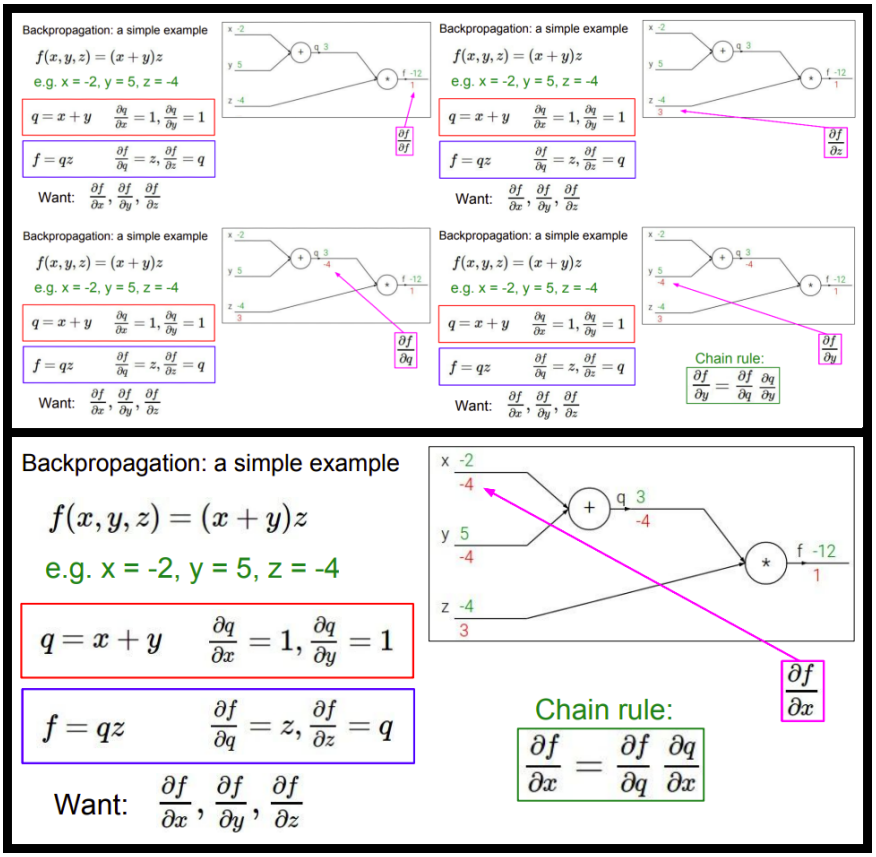


Figura 6.3.: Como podemos ver en la imagen superior, en primer lugar se renombra la salida de la operación $x + y$ por q, de manera que $f = qz$. Tras esto se empiezan a calcular las derivadas parciales correspondientes desde el final hacia la entrada, aplicando cuando sea necesario la regla de la cadena hasta obtener la derivada de cada nodo en la imagen de abajo. Las imágenes han sido extraidas de [Fei17]

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:

- **Conmutatividad**:
$$f*g = g*f$$
- **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**.

6.4.1. 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 de menores dimensiones será el 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 de salida de los mapas de activación pues la convolución es una operación que reduce la dimensionalidad siempre.

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:

$$Output = (W - F + 2P)/S + 1$$

Generalmente la salida de cada capa convolucional supone la entrada a una función de activación como las que se describieron en Subsección 6.3.1. Normalmente se utiliza la función ReLU, aunque también es común el uso de Leaky ReLU.

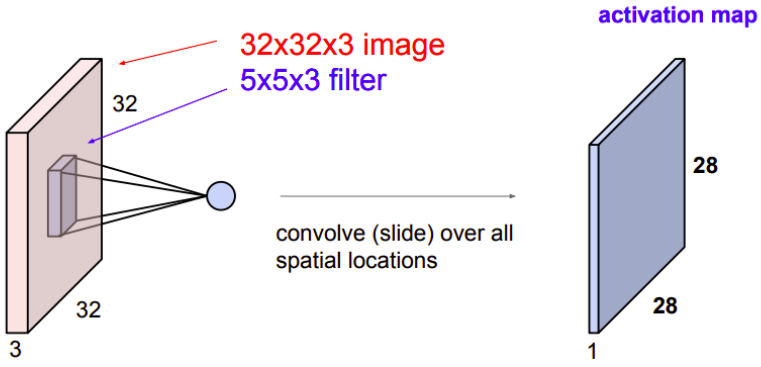


Figura 6.4.: Ejemplo de cálculo de mapa de activación en una capa convolucional para un determinado volumen de entrada. El parámetro depth nos dice la cantidad de mapas de activación generamos para el volumen de entrada, o dicho de otro modo, el número de filtros que aplicamos. La imagen ha sido extraida de [Fei17]

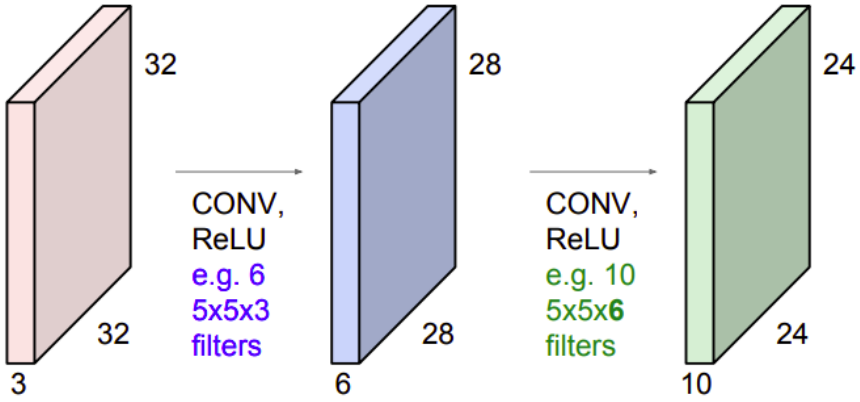


Figura 6.5.: Sucesión de varias capas convolucionales + ReLU que describen la estructura básica de una CNN. Cabe destacar como la produndidad de los filtros es siemore la misma que la del volumen de entrada, de acuerdo a lo que hemos dicho anteriormente. La imagen han sido extraida de [Fei17]

6.4.2. Capa de Pooling

En las CNN es normal instertar 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 2×2 con un stride de 2 que reduce a la mitad la dimensión del volumen de entrada y mantiene la profundad.

El tipo de operación que se realiza con el filtro 2×2 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 realzar 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.

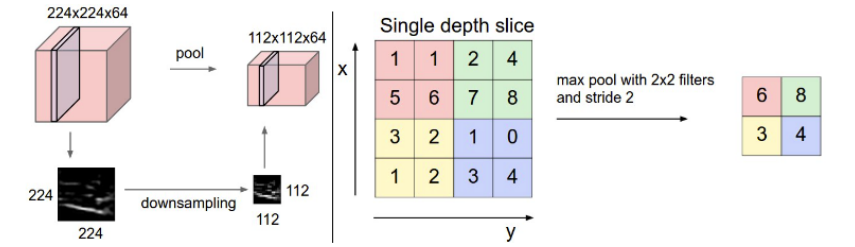


Figura 6.6.: Ejemplo de capa de pooling usando la operación del máximo. La imagen han sido extraida de [Fei17]

6.4.3. Capa Totalmente Conectada (Fully Conected)

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 actua 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.

6.4.4. 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 regularización.

6.4.5. 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 Back-propagation. 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.

6.4.6. 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 entrenan 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 calculado los gradientes de cada unidad con lo que se conoce como **backward pass** aplicando el algoritmo de back-propagation.
- 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...)

6.4.7. 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].

6.4.7.1. 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.

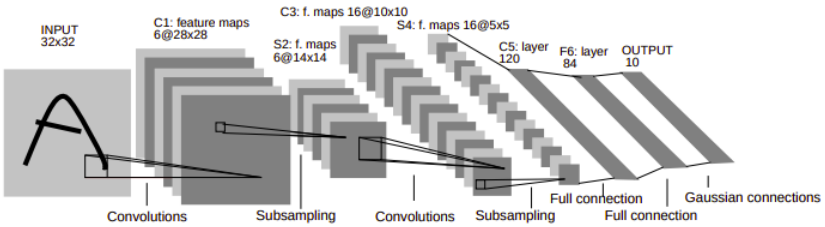


Figura 6.7.: Arquitectura de la red LeNet. Como podemos observar tiene capas convolucionales, capas de average pooling y capas totalmente conectadas al final. Imagen extraida de [LBBH98].

Algunas de sus características eran:

- Su arquitectura consiste en:

  $$INPUT \rightarrow CONV \rightarrow AVG\ POOL \rightarrow CONV \rightarrow AVG\ POOL \rightarrow FC \rightarrow FC \rightarrow OUTPUT.$$

- Las imágenes de entrada eran de dimensión 32×32 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.

6.4.7.2. 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 surgión la red AlexNet [KSH12].

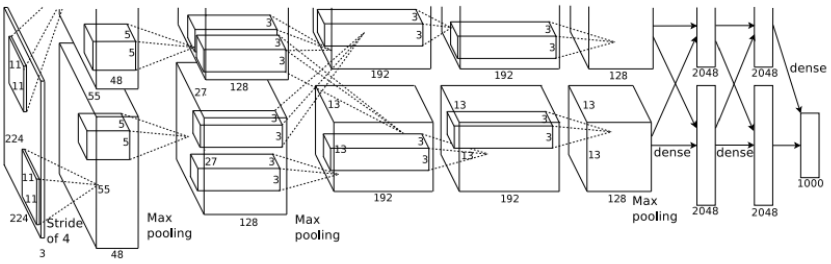


Figura 6.8.: Arquitectura de la red AlexNet. Destaca que parece estar *"partida"* en dos mitades, esto es porque por primera vez se usaron las GPUs para entrenar la red de manera que una GPU realizaba la parte superior de la arquitectura y otra la parte inferior. Imagen extraida de [KSH12].

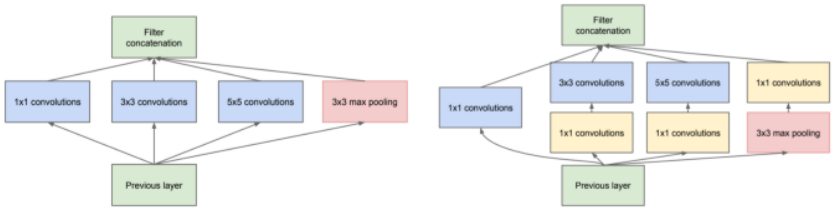
Destacan las siguientes propiedades:

- 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 del entrenamiento de acuerdo a una probabilida del 50 %. También se usó **data augmentation** que explicaremos en Subsección 6.6.1.
- El modelo ganó la competición ImageNet en 2012 con una diferencia en precisión del 11 % con respecto al algoritmo que quedó en segundo lugar.

6.4.7.3. 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 profundas se realicen simultáneamente múltiples convoluciones en una sola capa. Con este modelo aparecen los primeros filtros de convolución 1×1 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.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figura 6.9.: Ejemplos de módulos Inception.

Con este nuevo módulo se desarrolló en 2015 la red GoogLeNet [SLJ+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 **golbal average pooling** mediante la cual reducían la dimensión de los tensores con convoluciones 1×1.

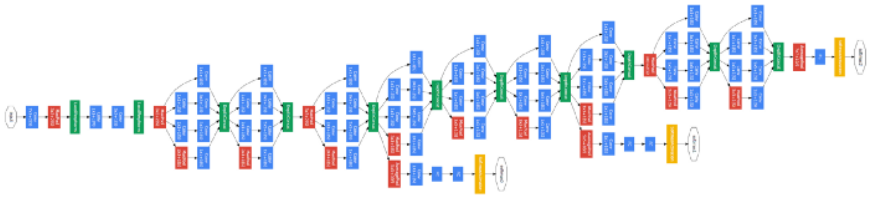


Figura 6.10.: Arquitectura de GoogLeNet usando módulos Inception.

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 1×1 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 multiclase.

6.4.7.4. 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 3×3. Esto ahorraba costes y permitía hacer una red más profunda [SZ14].

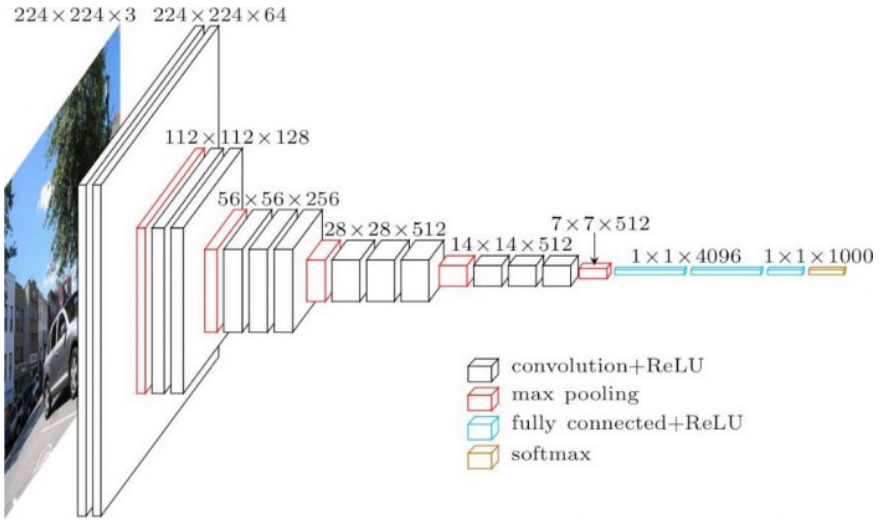


Figura 6.11.: Arquitectura de VGG-16.

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.

6.4.7.5. 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**.

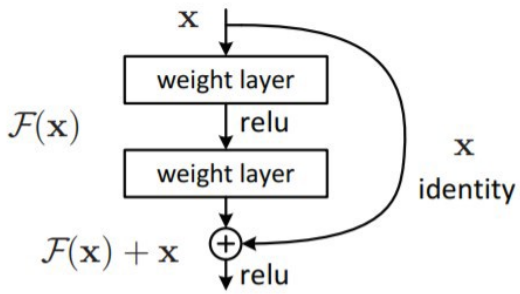


Figura 6.12.: Bloque residual de una ResNet. Como vemos, se suma el tensor x con el tensor $\mathcal{F}(x)$ que surge unas etapas después. Imagen extraida de [HZRS16].

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.

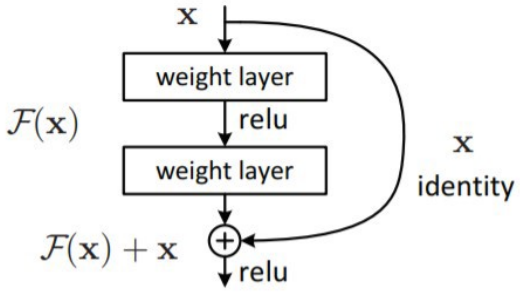


Figura 6.13.: Resumen de los ganadores del concurso ILSVRC hasta 2015 con la aparición de Resnet. Imagen extraida de [Fei17].

6.5. 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**.

6.5.1. 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*.

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**.

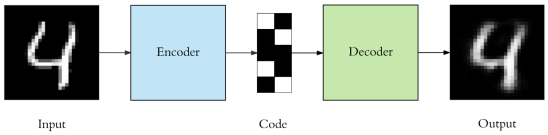


Figura 6.14.: Esquema de un Autoencoder básico. Imagen extraida de [Der17].

6.5.2. Evolución de los Autoencoders

6.5.2.1. 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 vector aleatorio y producir como salida una imagen de las mismas dimensiones que la de entrada que siga la distribución de probabilidad de las imágenes del dataset a nivel de **pixel**. Esta red tiene la estructura de un **Autoencoder**.

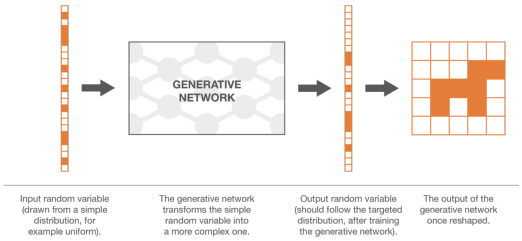


Figura 6.15.: Ejemplo de una Generative Network que pretende aprender la distribución de probabilidad de un conjunto de imágenes de perros. Imagen extraida de [Der17].

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.

6.5.2.2. 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.

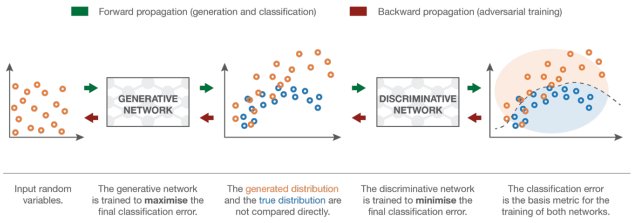


Figura 6.16.: Resumen del proceso de entrenamiento de una GAN. Imagen extraida de [Der17].

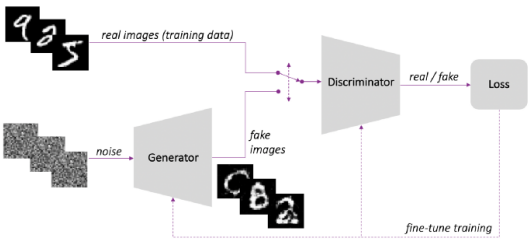


Figura 6.17.: Ejemplo de la arquitectura de una GAN para generar imágenes de dígitos manuscritos. Imagen extraida de [Ras20]

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.

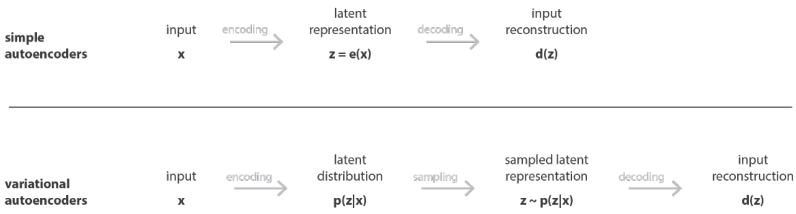


Figura 6.18.: Diferencia en el tratamiento de los datos por parte del encoder en una VAE y un Autoencoder clásico. Imagen extraida de [Roc19b].

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 regulariazante se incluye en la función de coste como podemos ver en Figura 6.19. Para ello utiliza la divergencia de *Kulback-Leibler* (KL) como medida entre la diferencia entre dos distribuciones.

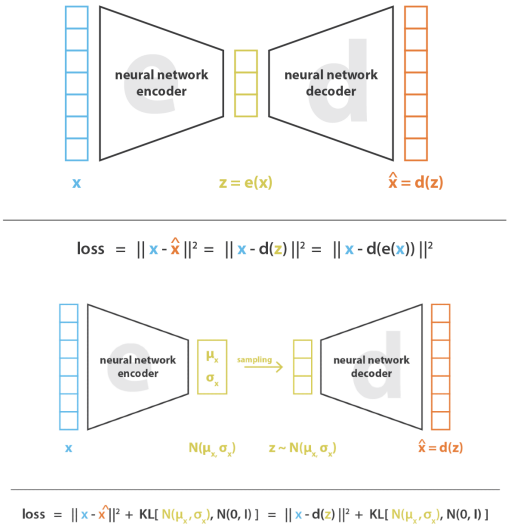


Figura 6.19.: Diferencia en la función de pérdida de un Autoencoder clásico (imagen superior) y una VAE (imagen inferior). Destacamos el término KL regularizante que pretende que las distribuciones de los datos sigan una normal estándar. Imagen extraida de [Roc19b].

6.5.2.3. 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 Discrimnante 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 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 ecoder.

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.

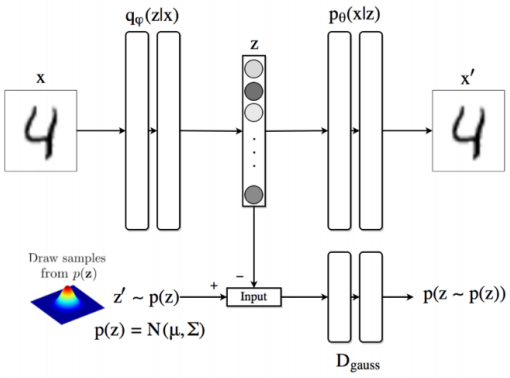


Figura 6.20.: Esquema de la arquitectura de un AAE. El Encoder sería la red a la que entra la imagen x, el vector z sería el vector latente, que sirve de entrada al Discriminador D_{gauss} y finalmente el vector z es la entrada del Decoder que reconstruye la imagen. Imagen extraida de [Ras20].

Esta arquitectura será la que emplearemos en nuestro proyecto, aunque presenta algunas variaciones que introduciremos más adelante.

6.6. 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.

6.6.1. 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 *oclusiones*. 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* (Figura 7.1) 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" ) )
```

Como podemos observar, 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 en la Figura 7.1 no guardan una relación muy estrecha con el problema de deteción de landmarks cefalométricos en problemas de antropología forense, por ello para descubrir el estado del arte en este campo nos vemos obligados a realizar otra consulta en scopus un poco más concreta y que nos permita conocer mejor las publicaciones más relevantes en este campo en los últimos años descartando los que no sean artículos relacionados con informática. La consulta que realizamos es la siguiente:

```
TITLE-ABS-KEY (
    (
        ( anthropology  OR  ( anthropology  AND  forensic ) )
        AND
        ( cephalometric  AND  ( landmarks  OR  keypoints ) )
    )
```

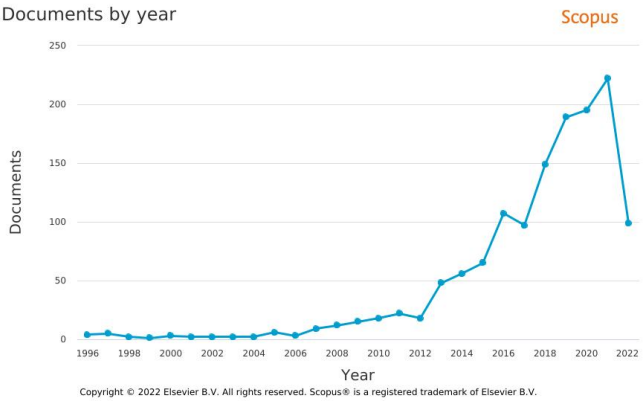


Figura 7.1.: Gráfica de publicaciones por año obtenida con la primera *keyword* el 14 de Julio de 2022. Destaca el notable incremento de papers a partir de 2012, año en que aparece la red AlexNet y comienza a ganar popularidad el Deep Learning en el tratamiento de imágenes.

```
OR

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

Con la 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. Podemos ver un gráfico de resultados de la búsqueda en Figura 7.2.

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:

7.1.1.1. Two different approaches to handle landmark location uncertainty in skull-face overlay: coevolution vs fuzzy landmarks

Se trata de un artículo pulicado en 2011 por Óscar Ibáñez et al [ICD11] en el cual abordan el problema de la superposición craneofacial diseñando un nuevo algoritmo basado en

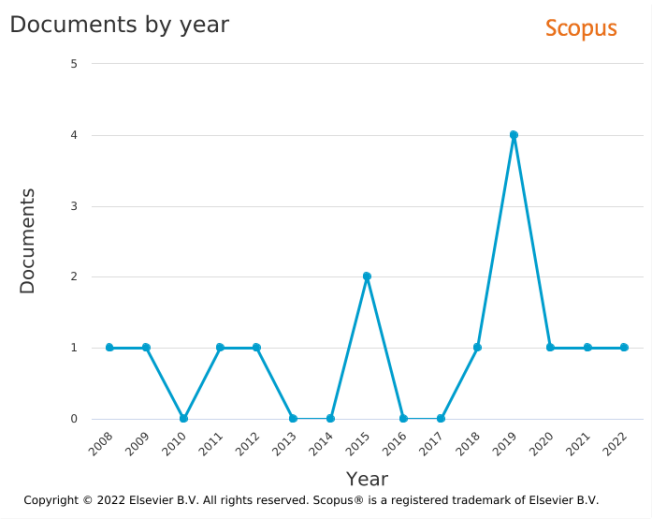


Figura 7.2.: Gráfica de publicaciones por año obtenida con la segunda *keyword* el 14 de Julio de 2022. La tendencia es a un artículo por año, aunque destaca el pico de tres artículos en 2015 y el de cuatro en 2019.

coevolución capaz de reducir el proceso de *Skull-face overlay* (SFO) que tradicionalmente podía llegar a tardar unas 24 horas y comparándolo con un algoritmo ya existente basado en landmarks imprecisos en el cual el antropólogo forense marca regiones de la imagen en las que se encuentra cada landmark (esta tarea no está automatizada en este algoritmo). Aunque no guarda una relación directa con el problema, el algoritmo propuesto realiza una identificación de landmarks cefalométricos en imágenes, por lo que hemos decidido incluirlo en el estudio.

Durante el proceso de SFO se dispone de un modelo 3D de un cráneo y de una imagen, de manera que se considera exitoso el proceso cuando se coloca el cráneo en la misma posición en que aparece en la imagen. Durante este proceso se deben tener en cuenta factores como la edad de la persona, el peso o las expresiones faciales, que añaden un grado más de dificultad a la tarea.

Generalmente se dispone de un conjunto de landmarks marcados tanto en la imagen como en el cráneo, y la tarea del SFO se reduce en calcular la transformación que permite llevar los landmarks del cráneo a ocupar la misma posición que en la imagen.

La alternativa al algoritmo basado en landmarks imprecisos es un algorimto coevolutivo en el cual el valor de la función de fitness de cada elemento depende de la de él mismo y otros individuos que pueden interaccionar de forma cooperativa o conflictiva. Adaptado al problema de SFO tendríamos dos poblaciones: el conjunto de parámetros que definen la transformación que se aplica al modelo 3D del cráneo y por otro lado las localizaciones de los landmarks cefalométricos. Ambas poblaciones colaboran para encontrar la mejor transformación posible y solucionar el SFO.

En el experimento se disponía de seis procesos distintos de SFO correspondientes a tres casos reales proporcionados por el laboratorio de Antropología Física de la Universidad de Granada en colaboración con la policía científica. Se disponía así de tres modelos 3D

de cráneos pertenecientes a personas desaparecidas junto con un dataset de seis imágenes. Las imágenes, a pesar de ser pocas, presentan una gran variedad en iluminación, posición (hay imágenes frontales y en 3/4) y con problemas de oclusión en algunos casos para los landmarks a causa del cabello. Por otro lado la calidad de las imágenes es muy variada, habiendo imágenes de gran resolución y otras de baja calidad. Finalmente se disponen cuatro imágenes para el caso de estudio 3 y una única imagen para el caso de estudio 1 y 2.

En el experimento, el algoritmo empleado para la técnica de encontrar landmarks imprecisos es CMA-ES, y se comparó con el algoritmo coevolutivo desarrollado. Las conclusiones fueron que el nuevo algoritmo reducía considerablemente los tiempos de ejecución del algoritmo basado en landmarks imprecisos y que realizaba una Localización de landmarks cefalométricos apropiada.

Por lo tanto podemos considerar este primer algoritmo coevolutivo como la primera aproximación a nuestro problema de detección automática de landmarks. A pesar de lo pobre que es el conjunto de datos de entrenamiento, se emplean imágenes que se encuentran en nuestro dataset y con diversas posturas y resolución de imagen. No obstante, la parte de detección de landmarks no es la principal de este artículo, pues aunque es una consecuencia del algoritmo que se desarrolla, se pretende realizar con el mayor éxito posible la fase SFO.

7.1.1.2. 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+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 como se pueden ver en la Figura 7.3. 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.

7.1.1.3. 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.

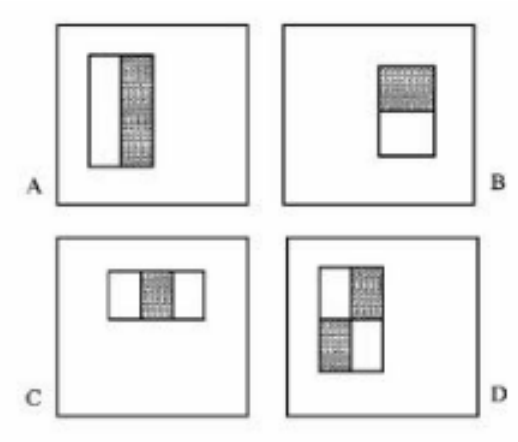


Figura 7.3.: Ejemplo de filtros empleados en el artículo [AIA+14] de donde procede esta imagen.

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 7.4, 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 7.5.

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 sobte modelos 3D de caras que evitan problemas de oclusión, mala iluminación o resolución.

7.1.1.4. 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 frontales a partir de técnicas de visión por computador y de métodos de aprendizaje supervisado [PLF+19].

El algoritmo tiene tres componentes:

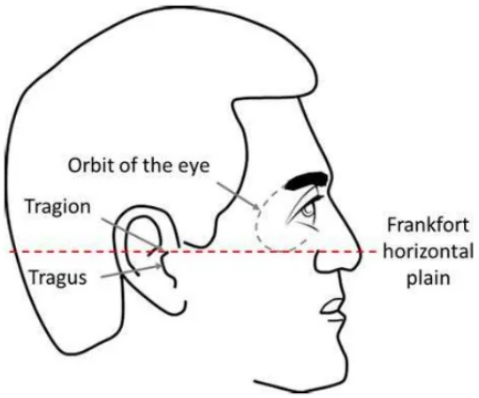


Figura 7.4.: Cara alineada con el plano horizontal de Frankfort. Imagen extraida de https://www.slideshare.net/NiharikaSupriya/cephalometrics-landmarkslines-and-planes-93890774

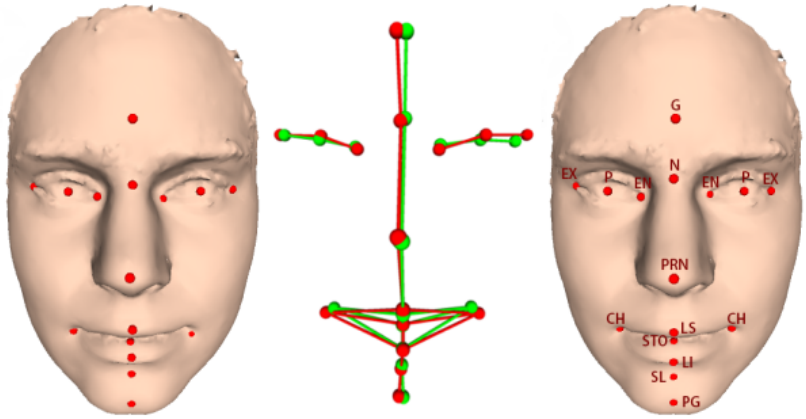


Figura 7.5.: Comparativa entre los landmarks marcados por el algoritmo (izquierda) y los marcados por un experto (derecha). Imagen extraida de [GFCS15].

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

7.1.1.5. 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 restalta 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.

7.1.2. Nuestra propuesta

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 desarrollado actualmente por la gran capacidad de las CNN para tareas de procesamiento automático de imágenes.

La principal diferencia entre este enfoque y el forense explicado en la sección anterior radica en el tipo de landmarks que se utilizan. Los utilizados en antropología forense son landmarks con justificación biológica, 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 7.6

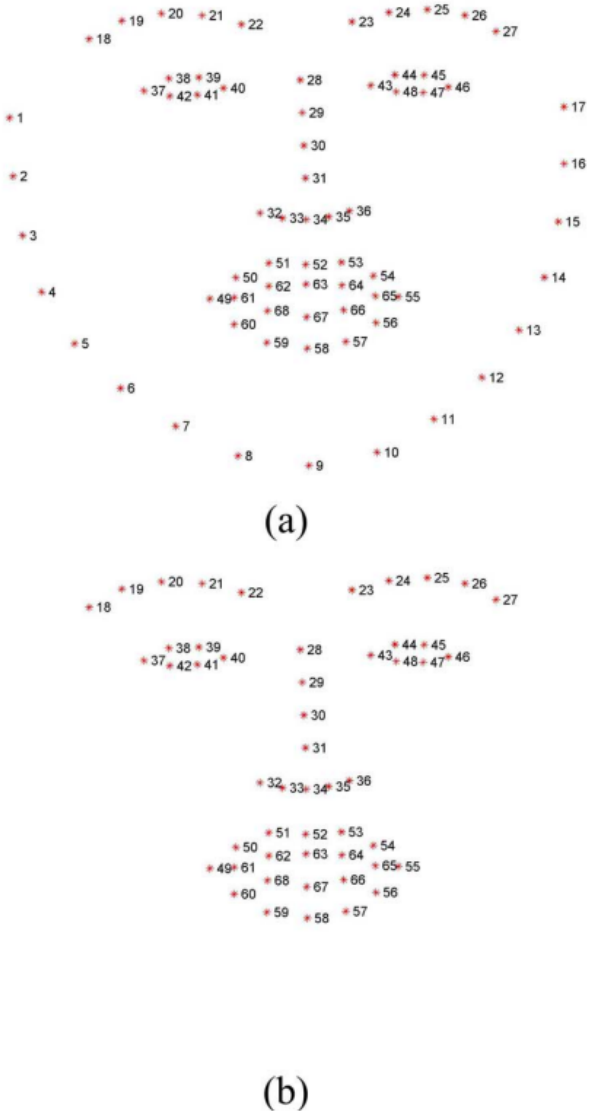


Figura 7.6.: Conjunto de landmarks anotados en el dataset 300W, en la imagen *a* contando el contorno del rostro son un total de 68 landmarks, en la imagen *b* son 51 en total. Imagen extraida de [STZP13].

- **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 7.7

[KWRB11]

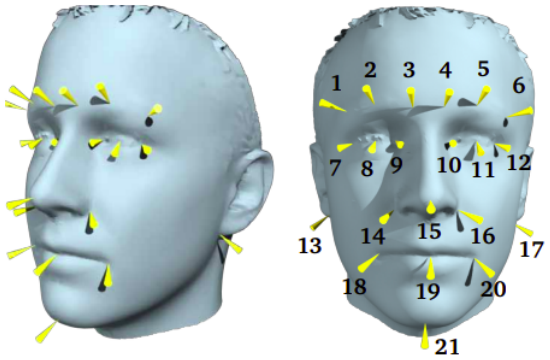


Figura 7.7.: Conjunto de landmarks anotados sobre un modelo 3D que emplea el dataset AFLW. Imagen extraida de [KWRB11].

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

8. Datos y Métricas

En este capítulo se van a presentar los datos de los que disponemos para resolver el problema principal de este trabajo (la identificación de landmarks cefalométricos mediante técnicas de few-shot learning), así como las principales métricas de error que se emplearán para estudiar la bondad de los resultados que se obtengan en futuros capítulos.

8.1. Datos del problema y framework empleado

8.1.1. 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. No se distribuye de forma equitativa el número de imágenes por sujeto, de manera que para algunos sujetos solo se dispone de una imagen mientras que otros disponen de varias. El sujeto con mayor número de imágenes tiene siete.
- La resolución de las imágenes también varía mucho, encontrando imágenes de alta calidad junto con otras con una muy baja resolución.
- Hay imágenes a color y en escla 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, aunque por regla general el número de landmarks en las imágenes es menor, como puede apreciarse en la Figura 8.1.
- En la Figura 8.1 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. La aparición en mayor o menor medida de cierto landmark en las imágenes nos sirve

  de indicador de si podrá ser o no aprendido por el modelo que usemos, de manera que los landmarks mencionados anteriormente, a causa del bajo número de ejemplos en los que aparecen puede ser más difícil que se aprendan.

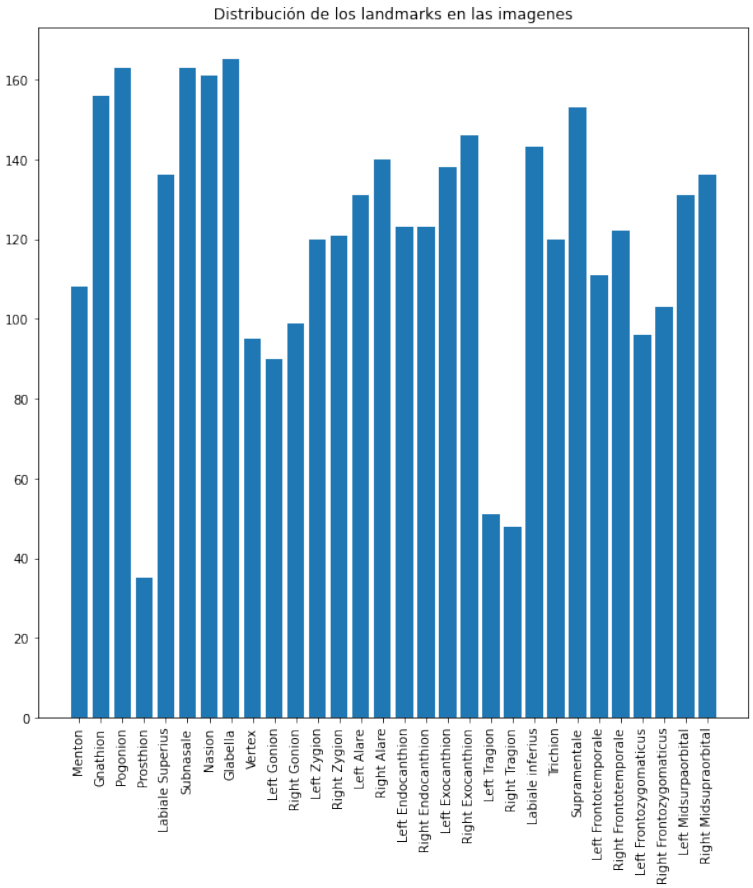


Figura 8.1.: Histograma con la aparición de cada tipo de landmark en las imágenes del dataset.

8.1.2. Red empleada: 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 del estilo que hay en las GAN) que puede predecir landmarks gracias a la incorporación de unas capas convolucionales intermedias denominadas *Interleaved Transfer Layer* en la etapa de reconstrucción y que explicaremos en profundidad más adelante.

Aplica un método *semi-supervisado* en el cual:

- Hay una primera fase de **aprendizaje no supervisado** dónde 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*.
- 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 encargadas de reconstruir los mapas de calor de cada landmark junto con la reconstrucción del rostro del paso previo.

- 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.

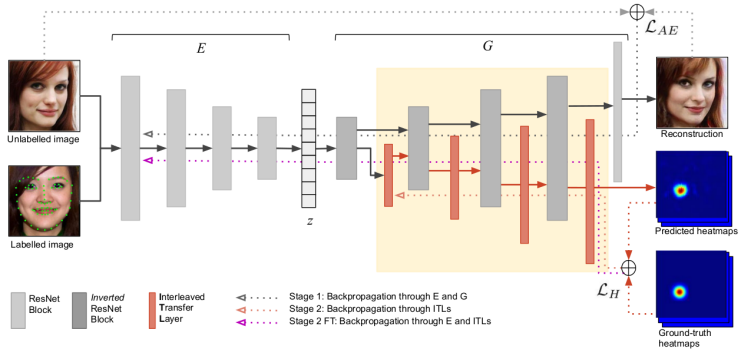


Figura 8.2.: Imagen resumen del framework 3FabRec. En ella podemos ver la estructura del *Adversarial Autoencoder*, dividido en un Encoder (región bajo la E) y un Generator (región bajo la G)

8.1.2.1. Arquitectura Adversarial Autoencoder

Para la construcción del *Adversarial Autoencoder* emplean:

- **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.

Para una mejor comprensión he realizado unos diagramas con la herramienta *diagrams.net*. En la Figura 8.3 podemos ver la estructura básica de los bloques de la ResNet-18.

Por otro lado en la Figura 8.4 podemos ver el paso de una imagen de entrada de tres canales y resolución 256×256 por el encoder.

En la Figura 8.5 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 8.6 Finalmente, para definir el autoencoder necesitamos un Discriminador, el cual que procure que los vectores del espacio vectorial latente sigan una determinada distribución. Dicha distribución será una normal multivariante estándar a la que vamos a añadirle 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. En la Figura 8.7 las redes neuronales que definen ambos discriminadores.

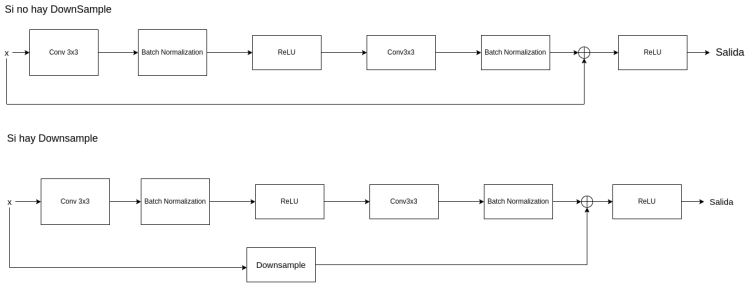


Figura 8.3.: Bloques básicos que utilza la red ResNet-18 en sus capas. Se trata de una sucesión clásica de Convolución 3x3 + Batch Normalization + ReLU que se repite dos veces. En el primer caso los filtros de convolución no reducen las dimensiones del tensor añadiendo un padding de 1. En el segundo caso se reduce la dimensión del tensor a la mitad tras la primera convolución y se manteiene 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.

8.1.2.2. Interleaved Transfer Layer (ITL)

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 8.6.

8.1.3. Función de pérdida

Para el entrenamiento de la red se emplean dos funciones de pérdida, la primera que presentaremos se emplea en el entrenamiento del *Adversarial Autoencoder*, en la parte no supervisada, y la segunda función de pérdida se empleará tanto en la parte de aprendizaje supervisado como en la de *finetuning* del Encoder.

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 λ_{enc} y λ_{adv} toman el valor 1.0 mientras que λ_{rec} y λcs 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*.

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

$$\mathcal{L}_H(ITL) = \mathbb{E}_{x\ p(x)}\left[\|H - ITL(a_1)\|_2\right] \tag{8.1}$$

Dónde a_1 serían los mapas de activación que genera el primer bloque de 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_2entre los *Heathmaps* de los lanmarks originales de la imagen x de entrada, y los predichos por las *ITLs*. Propagando el error por los pesos de las *ITLS* solamente.

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 128×128 en el caso de que las imágenes de entrada sean de 256×256. 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 eliminen 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.

8.1.4. Proceso de entrenamiento de la red

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 2×10^{-5}.

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

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 reescala la imagen hasta tener un tamaño de 256×256.

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

Tras esto se entrenan las cuatro *ITLs*. A los datos de entrada se les aplican técnicas de *data-augmentation* como rotaciones, traslaciones, reescalados y oclusiones. Para esta etapa se usa también un optimizador Adam con un *learning rate* de 0.001 y los mismos valores para β_1 y β_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 2×10^{-5} y cambiando el valor de $\beta_1 = 0.9$.

8.1.5. Bases de datos usadas por el framework

Para el entrenamiento no supervisado 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 mayor 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 no 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.

8.2. Metricas

A conitnuación mostraremos las métricas empleadas para estudiar la bondad de los resultados. Algunas de estas métricas ya se emplearon en la funciones de coste del apartado anterior.

8.2.1. SSIM

Se trata del *structural similarity index* (SSIM) y da una medida de *similaridad* entre dos imágenes [WBSS04], 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 σ_x, σ_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 σ_{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 α, β, γ 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 31×31.

8.2.2. Average pixel error

Se trata de la función de coste **L1** que se aplica a la imagen original y la reconstruida y nos porporciona una medida del error de reconstrucción a nivel de pixel. Este error se obtiene para cada imagen y luego se devuelve el error promedio, por eso se denomina *Average pixel error*.

8.2.3. MSE

Se trata de la función de coste **L2** usada para medir la diferencia entre los mapas de calor pertenecientes a los landmarks reales y los mapas de calor de los landmarks predichos.Su expresión es (8.1)

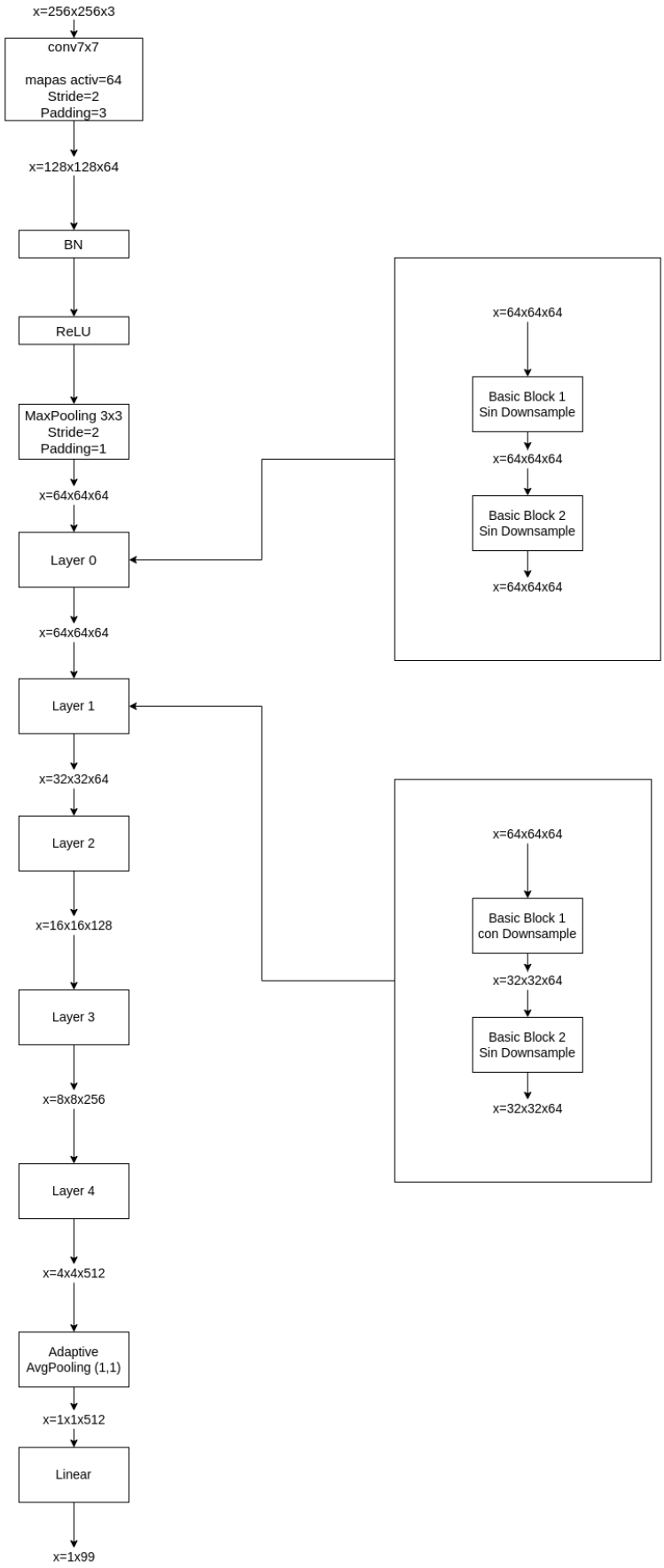


Figura 8.4.: Ejemplo de paso de una imagen a través del Encoder. Cabe destacar que a partir de la Layer 1, todos los bloques tienen downsample.

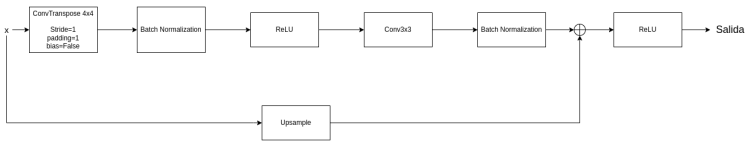


Figura 8.5.: En primer lugar se aplica una convolución transpuesta que duplica las dimensiones del tensor de entrada y tras esto se sigue la misma estructura que en el bloque básico de la ResNet-50, la segunda convolución 3×3 mantiene las dimensiones. Como consecuencia, para sumar el tensor de entrada con la salida del bloque se aumentan las dimensiones de este.

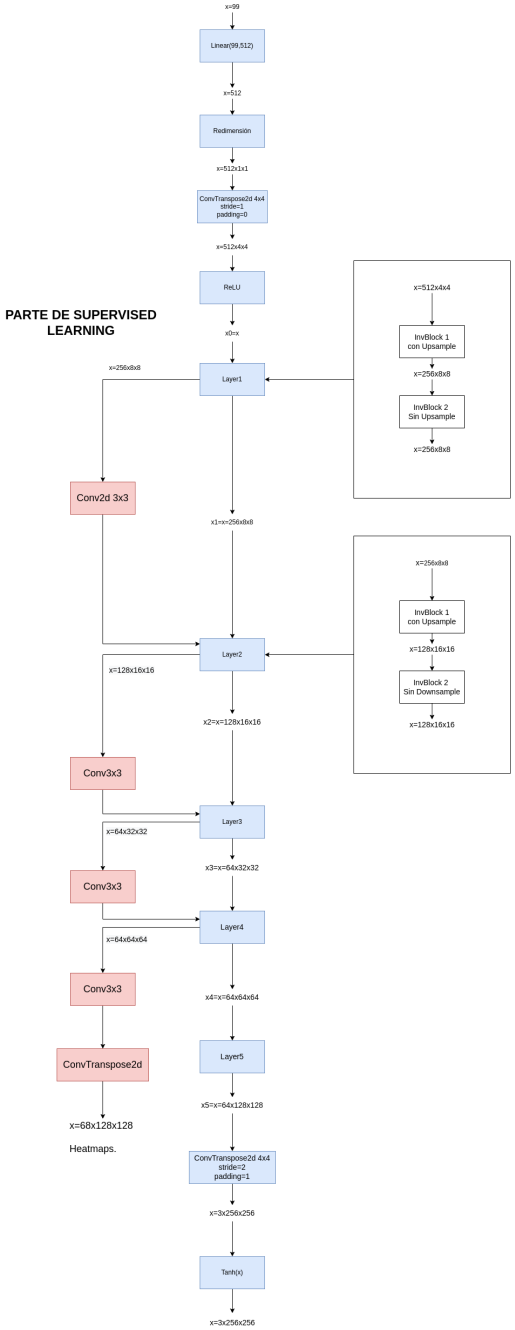


Figura 8.6.: Ejemplo del paso de un vector de 99 dimensiones por el generador hasta reconstruirse la imagen de dimensiones $256 \times 256 \times 3$. La parte correspondiente al aprendizaje supervisado es la de los cuadrados azules, los cuadrados rojos corresponden a las *ITLS* de la parte supervisada que se intercalan entre cada dos capas y dan como resultado los mapas de calor de los landmarks predichos.

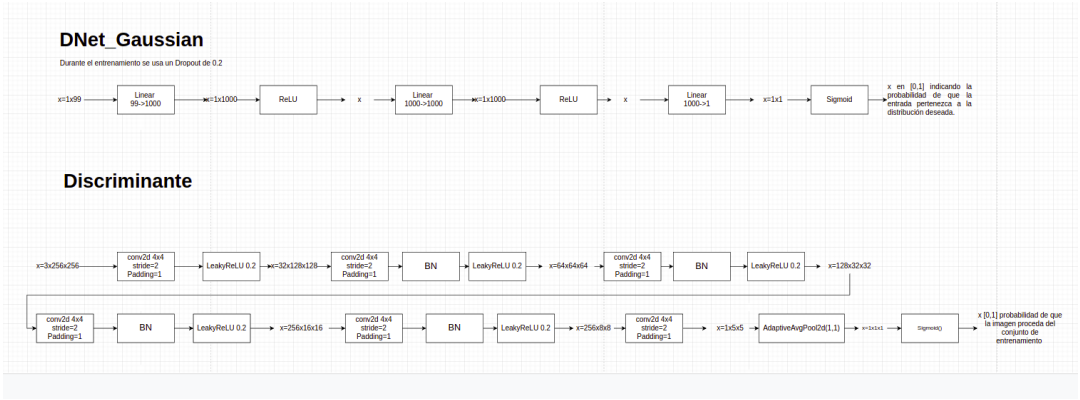


Figura 8.7.: En la imagen superior vemos el discriminante que se emplea para los vectores producidos por el Encoder y en la imagen inferior vemos el discriminante que se emplea para las imágenes generadas por el Generador. En ambos casos se da como salida un valor entre 0 y 1 que hace referencia a la 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. Planificación e implementación

10. Experimentación

11. Conclusiones y Trabajos Futuros

A. Primer apéndice

Los apéndices son opcionales.
Archivo: `apendices/apendice01.tex`

Glosario

La inclusión de un glosario es opcional.
Archivo: `glosario.tex`

$\mathbb{R}$ Conjunto de números reales.

$\mathbb{C}$ Conjunto de números complejos.

$\mathbb{Z}$ Conjunto de números enteros.

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

[AIA+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 págs. 74 and 75]

[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 pág. 82]

[Der17] Arden Dertat. Applied deep learning. = https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798,, 2017. [Citado en págs. 65, 66, and 67]

[Fei17] Fei-Fei Li, Justin Johnson, Serena Young , Stanford University. cs231n. = http://cs231n.stanford.edu/2017/syllabus.html,, 2017. [Citado en págs. 56, 57, 59, 60, and 65]

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. [Citado en pág. 54]

[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. 74 and 76]

[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 networ architectures. = https://medium.com/the-pen-point/evolution-of-convolutional-neural-network-architectures-6b90d067e403,, 2020. [Citado en pág. 61]

[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. 47]

[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. 77]

[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 pág. 64]

[ICD11] Óscar Ibáñez, Óscar Cordón, and Sergio Damas. Two different approaches to handle landmark location uncertainty in skull-face overlay: coevolution vs fuzzy landmarks. In *Proceedings of the 7th conference of the European Society for Fuzzy Logic and Technology*, pages 334–341. Atlantis Press, 2011. [Citado en pág. 72]

[J.B12] J.Bruna. Operators commuting with diffeomorphisms. *CMAP Tech. REport*, 2012. [Citado en pág. 22]

[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. 62]

[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. 79]

[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 pág. 61]

[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. 27]

[Mal00] 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, and 30]

[PJDoM06] University of Iowa Palle Jorgensen Department of Mathematics. Image decomposition using haar wavelet. = https:/https://homepage.divms.uiowa.edu/ jorgen/Haar.html/homepage.divms.uiowa.edu/ jorgen/Haar.html,, 2006. [Citado en pág. 15]

[PLF+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. 75]

[QV18] Stephen Quinn and Igor E Verbitsky. A sublinear version of schur's lemma and elliptic pde. *Analysis & PDE*, 11(2):439–466, 2018. [Citado en pág. 38]

[Ras20] Fathy Rashad. Adversarial auto encoder (aae). =https://medium.com/vitrox-publication/adversarial-auto-encoder-aae-a3fc86f71758,, 2020. [Citado en págs. 67 and 69]

[Roc19a] Joseph Rocca. Understanding generative adversarial networks (gans). = https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29,, 2019. [Citado en pág. 66]

[Roc19b] Joseph Rocca. Understanding variational autoencoders (vaes). = https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73,, 2019. [Citado en págs. 67 and 68]

[Ros88] Azriel Rosenfeld. Computer vision: basic principles. *Proceedings of the IEEE*, 76(8):863–868, 1988. [Citado en pág. 53]

[SLJ+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. 63]

[SSA17] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017. [Citado en pág. 54]

[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 pág. 78]

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. [Citado en pág. 64]

[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]

[TY05] Alain Trouvé and Laurent Younes. Local geometry of deformable templates. *SIAM Journal on Mathematical Analysis*, 37(1):17–59, 2005. [Citado en pág. 6]

[WBSS04] 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. 86]

[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,. [Citado en pág. 17]