RECONOCIMIENTO FACIAL EN IMÁGENES

PABLO ANDRÉS PASTORE

Trabajo Especial de la Licenciatura en Ciencias de la Computación

Facultad de Matemática, Astronomía, Física y Computación
Universidad Nacional de Córdoba

Director: Jorge Sánchez

Mayo 2018

A mi familia, por su cariño y apoyo incondicional, pilar fundamental
que hoy me ha permitido culminar esta etapa.

AGRADECIMIENTOS

Agradezco a mi director, Jorge Sánchez, por darme la posibilidad de realizar este trabajo con él. Por su paciencia, predisposición, y por todo lo que me ha enseñado en el trayecto.

A Leandro Lichtensztein y Agustin Caverzasi, por su confianza y apoyo para emprender este trabajo.

A Melina, por estar en todo momento a mi lado, aconsejando y acompañándome, gracias por motivarme siempre a dar un paso más hacia adelante.

A mis amigos, atentos y mostrándose cercanos, con ustedes siempre encontré un espacio para pasarla bien, fundamental para recargar energías en los momentos más difíciles.

RESUMEN

En este trabajo se abordará el problema de reconocimiento facial en imágenes utilizando redes neuronales convolucionales. El éxito actual en los resultados obtenidos por este tipo de modelos yace principalmente en la posibilidad de contar con grandes volúmenes de datos anotados para su entrenamiento. Con respecto al reconocimiento facial, la disponibilidad de bases de datos públicas para este tipo de problemas es escasa, restringiendo los avances en el área de los últimos años a grandes empresas como Facebook, Google, Baidu, etc.

En nuestros experimentos, tomaremos como punto de partida una red neuronal convolucional entrenada sobre una base de datos de rostros varios órdenes de magnitud menor a aquellas utilizadas comúnmente por el estado del arte. Usaremos vectores de características obtenidos con este modelo para entrenar un mapeo bilineal mediante una función de costo conocida como *Triplet Loss*. El objetivo del modelo final es obtener resultados cercanos al estado del arte, pero, utilizando un conjunto de datos de entrenamiento reducido.

ABSTRACT

In this work we address the problem of facial recognition on images using convolutional neural networks. Successful results obtained by these models relies mainly on the availability of large amounts of manually annotated data for training. For the task of facial recognition, the lack of large-scale publicly available datasets has restricted most advances in the field to big companies like Facebook, Google, Baidu, etc.

In our experiments, we start from a simple convolutional neural network trained on a standard facial recognition dataset several orders of magnitude smaller than those used by the state-of-the-art. We use features extracted from this model to train a bilinear map using a cost function known in the literature as *Triplet Loss*. The goal of such model is to get results as close as possible to those obtained by state-of-the-art models but using a much smaller training set.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE CUADROS

1

INTRODUCCIÓN

Un sistema de reconocimiento facial es una aplicación de computadora capaz de *verificar* o *identificar* una persona a partir de su rostro en una imagen digital o un fotograma de video. En la tarea de verificación facial se lleva a cabo una comparación uno a uno entre dos rostros para entonces confirmar si corresponden a la misma identidad o no. Por otro lado, la identificación facial realiza una comparación uno a varios entre un rostro, cuya identidad desconocemos, contra una base de datos de rostros conocidos para determinar la identidad de la nueva imagen.

Los sistemas de reconocimiento facial han ganando importancia en los últimos años y existen principalmente dos razones. La primera de estas es su uso en un amplio rango de aplicaciones comerciales y de seguridad. Tenemos como ejemplo el caso de Australia, que ha utilizado, con éxito, sistemas de reconocimiento facial en aeropuertos. Este país cuenta con un software para controles fronterizos llamado *SmartGate*, el mismo compara el rostro de un individuo con la imagen en el microchip de su pasaporte electrónico para verificar que la persona sea el dueño auténtico de este. Otro ejemplo, en donde el reconocimiento facial se ha adoptado para uso masivo, es en su aplicación como sistema de seguridad biométrico en dispositivos móviles, pues con el tiempo estos dispositivos se han vuelto una herramienta fundamental para las personas, quienes almacenan en ellos todo tipo de datos privados e información valiosa y sensible; surgiendo entonces la necesidad de contar con un sistema de seguridad que proteja el dispositivo de intrusos. Las *contraseñas alfanuméricas* o los *patrones de bloqueo* pueden ser olvidados e incluso ser tediosos de usar, entonces, contando con un sistema capaz de proteger el dispositivo utilizando la fotografía del rostro de su usuario simplifica enormemente la tarea. Pero claro, todo esto no seria posible si no se contase con un sistema preciso, siendo este el segundo punto clave en la importancia adquirida recientemente en el software de reconocimiento facial, pues principalmente gracias a los avances en algoritmos de aprendizaje automático por computadoras y en especial al uso de redes neuronales convolucionales se han obtenido sistemas capaces de identificar rostros con una precisión mayor a la de un humano [15]. Cabe destacar que su efectividad no es el único factor atractivo. En un comienzo, los sistemas de reconocimiento facial requerían de intervención humana para seleccionar regiones de interés en una imagen facial (tales como ojos, nariz, labios, etcétera). Actualmente se ha avanzado a sistemas automatizados que no requieren intervención humana y per-

miten comparar miles (y hasta millones) de rostros humanos en sólo segundos.

Para ser capaces de realizar estas comparaciones de identidad se necesita primero transformar la imagen del rostro de una persona en una representación numérica de este; comúnmente denominado *vector de características*, este contendrá valores que describan de cierta manera la información procesada. En el caso de reconocimiento facial, un vector de características posible seria la secuencia de valores de intensidad correspondientes a cada píxel que compone la imagen del rostro. El problema con esta descripción es que seria poco robusta, por ejemplo, respecto a cambios de iluminación en la imagen (producirían grandes alteraciones en los valores de los píxeles en esta). Como veremos mas adelante, para generar mejores representaciones numéricas a partir de imágenes emplearemos técnicas de aprendizaje automático, mas específicamente, en este trabajo utilizaremos algoritmos de *redes neuronales convolucionales* en conjunto con un algoritmo conocido como *triplet loss*.

Una vez generado el vector de características podemos proceder a realizar la identificación o verificación facial. Para esta tarea necesitamos contar con una función que compare vectores numéricos y retorne un valor de distancia entre estos. En el caso de verificación facial, nuestra función de distancia es aplicada entre dos vectores, diremos que se trata de la misma persona si obtenemos un valor de distancia pequeño, idealmente cero; por otro lado, diremos que serán personas distintas si su distancia es mayor a algún valor predefinido. Con respecto al reconocimiento facial, en primera instancia tendremos generada una base de datos de vectores (previamente computados), cada uno de estos con alguna identidad asignada. La identidad del nuevo vector de características obtenido sera la misma que la del vector mas cercano en nuestra base de datos.

Finalmente, tal como se muestra en la Figura 1.1, el proceso de reconocimiento facial se puede dividir en cuatro componentes principales:

1. Detección de rostros: se debe decidir en qué partes de una imagen está presente una cara.

2. Normalización: una vez detectado el rostro, este debe ser estandarizado en términos de tamaño, pose, iluminación, etcétera, relativo a las imágenes de nuestra base de datos.

3. Extracción de características: en esta etapa se genera una representación numérica de las características principales de una cara.

4. Reconocimiento: usando las representaciones obtenidas en el paso anterior y alguna función de distancia, se procede a identificar o verificar el vector de características obtenido previamente.

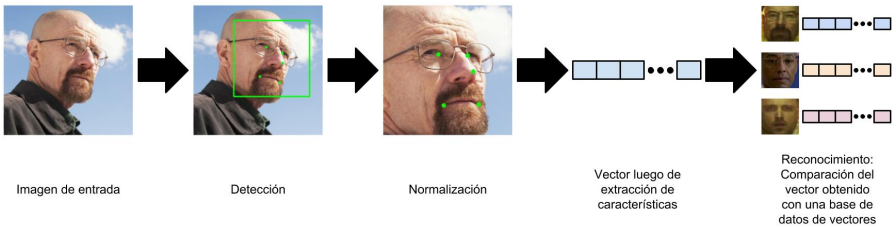

Figura 1.1: Etapas en el proceso de reconocimiento facial.

Este trabajo esta organizado de la siguiente manera: en el Capítulo 2 se introducirán los conceptos teóricos necesarios para comprender las técnicas y algoritmos que se emplearán. En el Capítulo 3 se abordara en profundidad el tópico de reconocimiento facial con redes neuronales convolucionales y triplet loss. A partir de aquí, en el Capitulo 4 se detallaran los experimentos realizados, mostrando sus resultados y comparándolos con aquellos obtenidos por otros métodos. Finalmente en el Capitulo 5 se presenta una conclusión y se plantean posibles trabajos a futuro en el área.

1.1 TRABAJOS PREVIOS

El primer método popular para reconocimiento facial fue Eigenface [33] propuesto en 1991. Este método mostraba cómo construir una función para transformar las imágenes de rostros de personas a un espacio de vectores propios (eigenvectores) donde los vectores correspondientes a imágenes de una misma persona sean más cercanos entre sí comparados con vectores de una persona distinta.

En 1997 surge Fisherfaces [1], en donde se proponen varias mejoras con respecto al método anterior con el objeto de dar mayor robustez ante efectos de iluminación en la imagen y variación en las expresiones faciales.

Luego de esto se propuso el uso de métodos para extracción de vectores de características tales como *SIFT*, siendo este robusto a cambios de escala, rotación, iluminación y puntos de vista; a partir de este método, un conjunto de vectores de características son generados en puntos fijos del rostro como ojos, nariz y boca; luego estos son concatenados o promediados para obtener un vector final [5].

A partir del año 2005 comienzan a surgir métodos de reconocimiento facial utilizando redes neuronales [4, 8, 18], se demuestra entonces que este nuevo algoritmo es más robusto en comparación a los utilizados previamente y que posee una gran capacidad para analizar en profundidad los datos y expresar características representativas de lo que se está percibiendo. A partir de este momento comienzan a surgir nuevas investigaciones utilizando redes neuronales pero ahora combinadas con otros métodos, tales como PCA [12], en un intento por mejorar los resultados obtenidos.

En 2012 las redes neuronales convolucionales pasaron a un primer plano obteniendo el primer puesto en la competencia ImageNet [23], donde este algoritmo logra resultados excepcionales sobre un conjunto de datos con un millón de imágenes correspondientes a mil clases, obteniendo una tasa de error de casi la mitad comparado con el mejor competidor. Este éxito llevó a una revolución en el área de la visión por computadoras, de la cual el reconocimiento facial no quedo exento. Es entonces en 2014 cuando se publica la red convolucional llamada "DeepFace" [31], reportando resultados de un 97,35 % de exactitud en la tarea verificación en el conjunto de datos LFW [13]. Luego de esto fueron surgiendo sucesivamente mejores modelos [29] hasta que en el mismo año se publica "DeepId2" [27] obteniendo un asombroso resultado de 99,15 % de exactitud en LFW.

El reconocimiento facial ha sido uno de los casos más famosos y estudiados [20, 24, 28], donde las redes convolucionales han obtenido resultados extraordinarios, superando incluso el desempeño humano [15]. En la actualidad el estado del arte reporta un resultado de 99,77 % en LFW, alcanzado por el gigante tecnológico Baidu [14].

2 MARCO TEÓRICO

Uno de los aspectos claves y de mayor importancia en la construcción de un sistema de reconocimiento facial es la etapa de generación de vectores de características a partir de imágenes faciales. Tal como se comentó previamente, utilizaremos técnicas de aprendizaje automático para obtener estos vectores numéricos.

En este capitulo daremos una introducción a la teoría del aprendizaje automático, las principales categorías en las que se encuadran los algoritmos en este área y como estos "aprenden" a realizar predicciones sobre un conjunto de datos dado, mostrando también algunos ejemplos de aplicaciones prácticas.

Una vez adquiridos estos conocimientos, presentaremos el algoritmo de redes neuronales, para luego avanzar hacia la teoría de redes neuronales convolucionales. Estas últimas serán utilizadas en nuestros experimentos para generar representaciones numéricas a partir de imágenes.

2.1 APRENDIZAJE AUTOMÁTICO

El aprendizaje automático es el subcampo de las ciencias de la computación y una rama de la inteligencia artificial que se basa en el estudio y el diseño de algoritmos que pueden hacer predicciones basándose en datos suministrados como ejemplos. Esto se logra utilizando distintos tipos de reglas estadísticas que permiten inferir patrones complejos en los datos proveídos, llamándole a este proceso *"entrenamiento"*. Los datos sobre los cuales nuestro modelo es entrenado se denomina comúnmente *conjunto de entrenamiento*, mientras que los datos en lo que realizamos predicciones pero no han sido analizados por el modelo durante el entrenamiento se conocen como *conjunto de evaluación*.

El aprendizaje automático tiene una amplia gama de aplicaciones y es clave en el desarrollo de algoritmos que resultarían imposibles de ser programados de manera explícita; ejemplos de estas aplicaciones son filtros de correo no deseado [2], reconocimiento del lenguaje escrito y oral [3, 17] o reconocimiento facial, tal como veremos en este trabajo.

Existen tres categorías principales de algoritmos que se diferencian en función del tipo de datos con los que son entrenados. Los principales son:

- Aprendizaje supervisado.

- Aprendizaje no supervisado.
- Aprendizaje por refuerzo.

APRENDIZAJE SUPERVISADO El algoritmo es entrenado con datos acompañados por una *"etiqueta"*, mostrando así que tipo de respuesta se espera obtener. Es a partir de estos datos que el algoritmo debe inferir patrones que le permitan decidir el tipo de respuesta que debe devolver al recibir nuevos valores de entrada no vistos previamente. Dentro del aprendizaje supervisado tenemos principalmente dos tipos de algoritmos conocidos como *clasificación* y *regresión*. En el caso de la clasificación se quiere realizar una predicción entre un numero finito de clases posibles como por ejemplo, un algoritmo que clasifique correos electrónicos en dos categorías, *"Correo deseado"* o *"Correo no deseado"*. Por otro lado, los algoritmos de regresión se encargan de predecir valores continuos, como por ejemplo, la estimación del valor de una casa basándonos en la cantidad de metros cuadrados de esta.

APRENDIZAJE NO SUPERVISADO Difiere del anterior en el sentido de que se proporcionan datos sin ninguna etiqueta, el algoritmo debe entonces inferir patrones en los datos que le permitan ordenarlos de la mejor manera posible. Supongamos por ejemplo que tenemos una base de datos con una gran cantidad de imágenes de lugares (calles, edificios, parques) y queremos agruparlas de acuerdo a la similitud entre estas. Este trabajo resultaría tedioso de hacer manualmente pero utilizando un algoritmo no supervisado podríamos dejar que este busque patrones que permitan relacionar y agrupar las fotografías de lugares similares. A este tipo de algoritmos se los llama de *Agrupamiento* (o *Clustering* por su denominación en inglés).

APRENDIZAJE POR REFUERZO Es un área que ha ido ganando especial importancia en los últimos años y se ha inspirado a partir de la psicología conductista. Estos algoritmos intentan determinar qué acciones debe escoger un agente de software en un entorno dado con el fin de maximizar alguna noción de "recompensa". Su información de entrada es la retroalimentación que obtiene como respuesta a sus acciones, convirtiéndolo así en un sistema que "aprende" a partir de ensayo-error. El ejemplo mas reciente en el área es el caso de AlphaGo Zero, desarrollado por Google DeepMind para jugar el juego de mesa Go [25]. Entre el 9 y 15 de Marzo de 2016, AlphaGo Zero se enfrento a Lee Sedol, reconocido campeón mundial de Go. Siendo este último derrotado por el software en 4 partidas de las 5 disputadas.

2.1.1 *Ejemplo: clasificación de imágenes*

Necesitamos abordar algunos conceptos básicos del aprendizaje automático que luego profundizaremos mas adelante en esta sección. Para

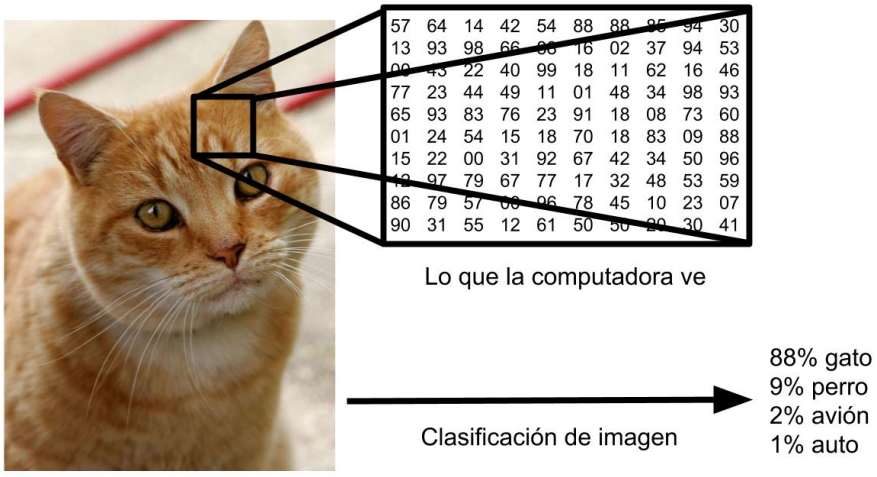

Figura 2.1: Clasificación de una imagen entre 4 categorías posibles, obtendremos un valor de probabilidad por cada clase.

esto nos enfocaremos en el aprendizaje supervisado, y en particular, analizaremos el problema de clasificación de imágenes.

Por ejemplo, supongamos que queremos obtener un modelo de clasificación el cual tome como entrada los valores de los píxeles de una imagen y devuelva probabilidades para cuatro categorías posibles: gato, perro, auto y avión; tal como se muestra en la Figura 2.1. Lo que estaríamos prediciendo es, la probabilidad de que la imagen pertenezca a alguna de esas categorías.

Para construir el clasificador de imágenes presentado, existen a grandes rasgos, tres componentes fundamentales que necesitamos analizar:

- Entrada
- Aprendizaje
- Evaluación

Nuestra *entrada* consistirá de un conjunto de K imágenes, cada una acompañada por una etiqueta, correspondiente a algunas de las 4 categorías posibles (gato, perro, auto o avión). Este será nuestro *conjunto de entrenamiento*. Es necesario destacar que, para una computadora, una imagen es representada como un vector numérico de tres dimensiones. Cada dimensión pertenece a un canal de color (rojo, azul o verde) y es una matriz de tamaño NxM, donde N es la altura y M el ancho de la imagen (en cantidad de píxeles). Nuestra imagen entonces esta compuesta por $NxMx3$ números representando el valor de cada píxel en la imagen, cada número es un entero que varia en un rango de 0 (negro) hasta 255 (blanco).

Supongamos ahora que tenemos un conjunto de entrenamiento de imágenes $x_i \epsilon \mathbb{R}^D$ (por simplicidad representaremos una imagen como un vector de D píxeles), cada una asociada a una etiqueta y_i. En este

ejemplo $i = 1...N$ e $y_i \epsilon \{1, 2, 3, 4\}$, donde el valor de y_i se corresponderá con el nombre de alguna de las clases a predecir, por ejemplo, $y_i = 1$ hará referencia a "gato", $y_i = 2$ se referirá a "perro" y así sucesivamente. Nuestro objetivo será entonces aproximar una función desconocida $f^* : \mathbb{R}^D \mapsto \{1, 2, 3, 4\}$, que mapea imágenes a su correspondiente clase.

Lo que haremos, será definir una *función de predicción* $f : \mathbb{R}^D \mapsto \mathbb{R}^4$, esta recibirá de entrada los valores de los píxeles de una imagen y retornará un valor de puntuación para cada una de las cuatro categorías posibles, tomando como categoría predicha aquella de mayor puntuación. Buscaremos luego, estimar los parámetros en f que nos permitan obtener la mejor aproximación posible de la función original f^*.

Una de las funciones de predicción mas simples es, la función lineal:

$$f(x_i; W, b) = Wx_i + b \tag{2.1}$$

En la ecuación 2.1 introducimos dos nuevos parámetros, la matriz W (de tamaño $4xD$) y el vector b (de tamaño $4x1$). Los parámetros en W se los denomina comúnmente *pesos* de nuestro modelo, mientras que a b se lo llama vector de *sesgo* (o *bias*, por su denominación en inglés). En la Figura 2.2 podemos ver una descripción gráfica de nuestra función de predicción, lo que esta ocurriendo en la multiplicación matricial Wx_i es la evaluación de 4 clasificadores en paralelo (uno por cada clase), donde cada clasificador es una fila de W. Por otro lado, b no interactúa con x_i pero si afectará en el valor de puntuación final asignado a cada clase. En la Figura 2.3 se muestra a modo representativo la "forma" de nuestro clasificador lineal para cuatro categorías.

Notar que, los valores (x_i, y_i) están dados y no podemos modificarlos, pero si podemos tomar el control sobre los valores de W y b. De hecho, en la sección 2.3 veremos como estos parámetros son estimados para intentar que el modelo se aproxime de la mejor manera posible a la función f^*, esto es, obtener como predicción la categoría correcta para cada imagen la mayor cantidad de veces posible. A este proceso se lo conoce como de *entrenamiento* o *aprendizaje* de nuestro clasificador.

Una vez generado el modelo final, nuestro último paso sera la *evaluación*. En esta etapa, utilizaremos el modelo para hacer predicciones sobre imágenes que este no ha visto durante el entrenamiento, a estas imágenes se las conoce como *conjunto de evaluación*. Luego, compararemos las predicciones obtenidas con las deseadas, obteniendo así una medida acerca del rendimiento del modelo.

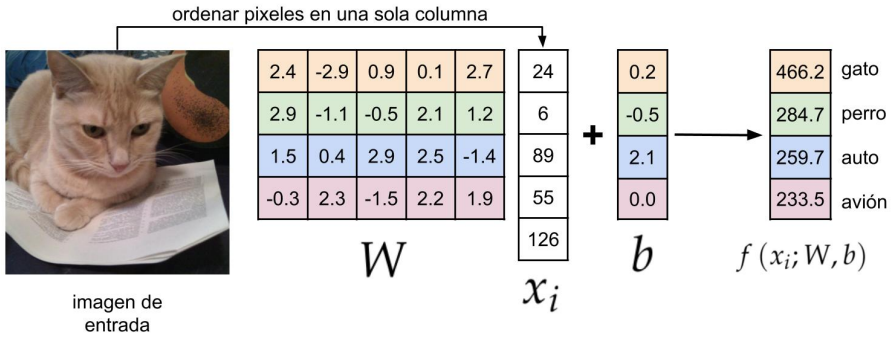

Figura 2.2: Ejemplo de función lineal retornando puntuaciones por cada clase para una imagen dada. Por motivos visuales asumimos que la imagen tiene sólo 5 píxeles.

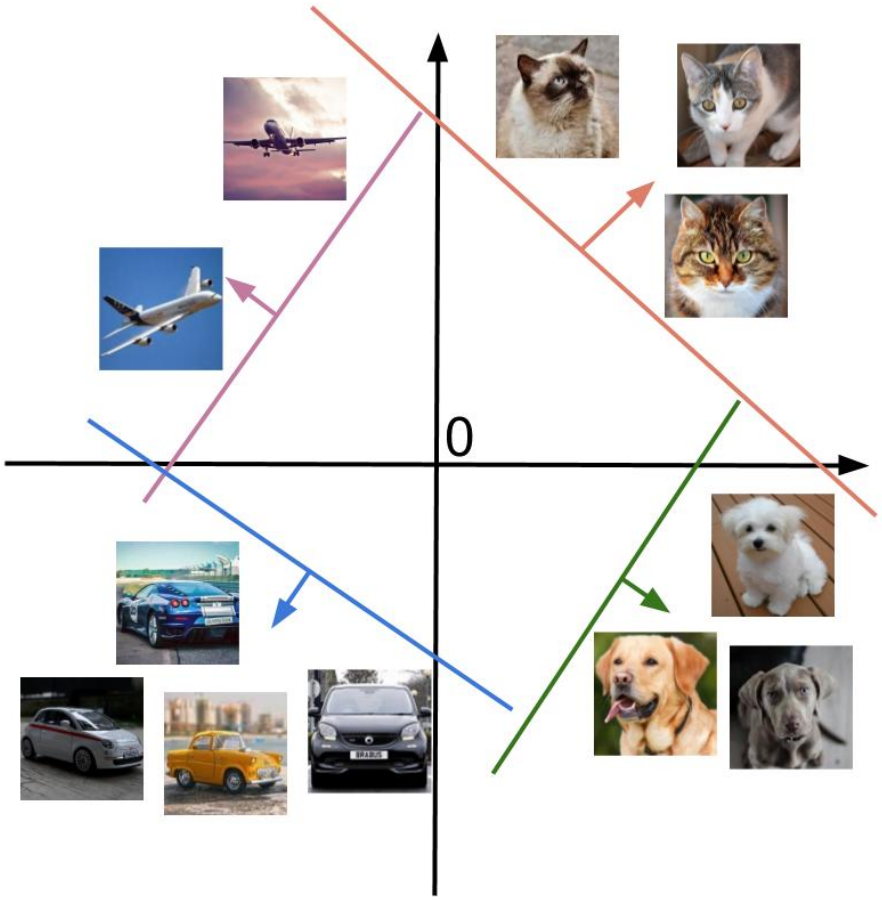

Figura 2.3: Representación en un espacio bidimensional de un clasificador lineal para cuatro categorías. Usando de ejemplo el clasificador de gatos (en naranja), la flecha naranja muestra la dirección de incremento de la puntuación para la clase gato. Todos los puntos a la derecha de la linea naranja tienen una puntuación positiva que se incrementa de manera lineal, por otro lado, aquellos puntos a la izquierda de la linea tienen una puntuación negativa.

2.2 REDES NEURONALES

Al igual que en el ejemplo presentado anteriormente, nuestra red neuronal intentará estimar los parámetros óptimos para una función $f(x; W, b)$ a partir de un conjunto de entrenamiento $\{(x, y)\}_1^N$. Cabe aclarar que, esta vez, f será una función no lineal, por ejemplo $f(x; W, b) = max(0, Wx + b)$. Es necesario destacar la importancia en el uso de funciones no lineales en este tipo de algoritmos, pues, también puede ocurrir que f es generada a través de la composición de varias funciones. Por ejemplo, podríamos tener tres funciones $f^{(1)}$, $f^{(2)}$ y $f^{(3)}$ conectadas en cadena, obteniendo así $f(x) = f^{(3)}\left(f^{(2)}\left(f^{(1)}(x)\right)\right)$. Pero si $f^{(1)}$, $f^{(2)}$ y $f^{(3)}$ fueran funciones lineales, componerlas daría como resultado una nueva función lineal.

Como veremos en las siguientes secciones, este esquema de composición de funciones es clave en los algoritmos de redes neuronales. Cada función que conectamos en cadena en nuestro modelo recibe el nombre de *capa*. Así, en el ejemplo anterior, $f(x)$ es una red neuronal de tres capas. A su vez, cada capa esta compuesta por *perceptrones*, cuya función sera el procesamiento de los datos y veremos en detalle a continuación.

2.2.1 *Perceptrón*

La unidad básica de procesamiento de una red neuronal son los perceptrones (o neuronas). Estos fueron desarrollados entre 1950 y 1960 por Rosenblatt, quien se inspiró en el trabajo de McCulloch y Pitts [16].

El perceptrón puede tomar uno o varios datos en binario como entrada, y retorna una única salida binaria. Para computar el resultado de salida de la neurona se aplican las siguientes fórmulas:

$$z(x) = \sum_{j=1}^{n} (w_j x_j) + b \tag{2.2}$$

$$a(z) = \begin{cases} 0 & z \leq 0 \\ 1 & c.c. \end{cases} \tag{2.3}$$

La ecuación en 2.2 simplemente multiplica cada peso con los valores de entrada y les agrega un valor de sesgo, algo similar a lo que ocurría en el clasificador lineal. La segunda ecuación (2.3), mas formalmente conocida como *función de activación*, define el valor de salida del perceptrón. En la Figura 2.4 podemos ver una descripción gráfica del funcionamiento de un perceptrón.

Volviendo a la variable de sesgo, esta regula qué tan fácil es obtener el valor 1 del perceptrón. Para un perceptrón con un valor de sesgo muy grande será fácil retornar el valor 1, asimismo si el sesgo es

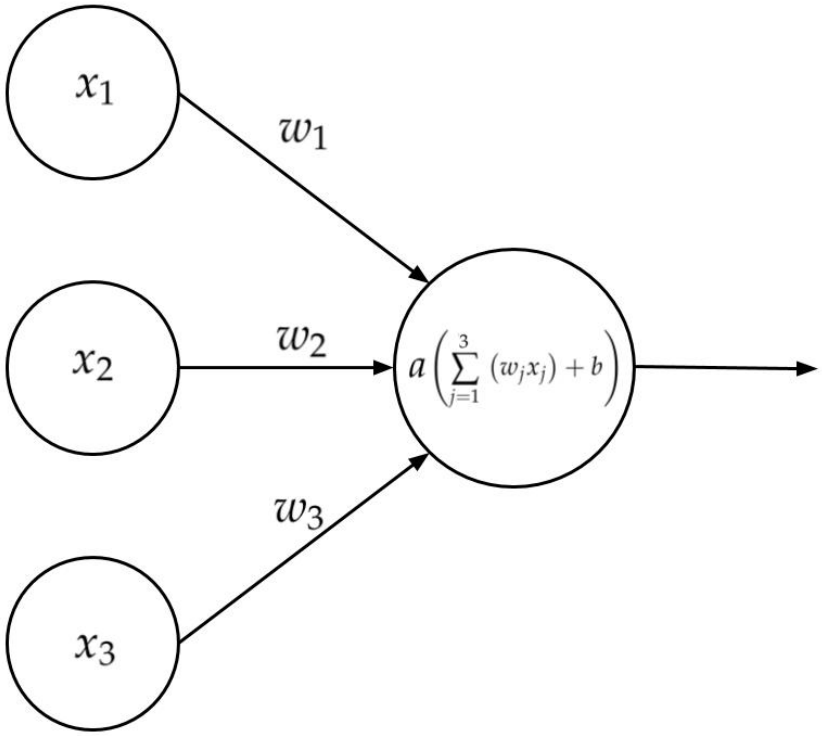

Figura 2.4: Diagrama de perceptrón con tres entradas (x_1, x_2, x_3) y una única salida.

un número muy negativo obtendremos siempre como resultado de salida el número 0.

Supongamos ahora que queremos utilizar el perceptrón para aprender la función de *Disyunción exclusiva* (o *XOR* por su denominación en inglés), esta es una operación entre dos valores binarios definida de la siguiente manera:

$$XOR\left(x_1, x_2\right) = \begin{cases} 1 & |x_1 - x_2| = 1 \\ 0 & c.c. \end{cases} \tag{2.4}$$

dados $(x_1, x_2)\,\epsilon X, X = \{(0,0),(0,1),(1,0),(1,1)\}$. La función XOR provee entonces la función objetivo $y = f^*(x)$ que intentaremos aprender, dados $x\epsilon X, y\epsilon\{0,1\}$; por otra parte, nuestro perceptrón provee una función $y = f(x; W, b)$, y a través de un proceso de entrenamiento buscaremos los parámetros en W y b que nos permitan aproximar de la mejor manera a f^* con f. Por lo visto previamente, dados $(w_1, w_2)\,\epsilon W, W\epsilon\mathbb{R}^2$ tenemos que $f(x; W, b) = a(w_1 x_1 + w_2 x_2 + b)$, siendo $w_1 x_1 + w_2 x_2 + b$ una función lineal. Si observamos la Figura 2.5, mostrando gráficamente la función XOR, vemos que las dos clases a predecir (0 y 1) no son separables utilizando una única función lineal y por ende, nuestro perceptrón fallará ajustando correctamente

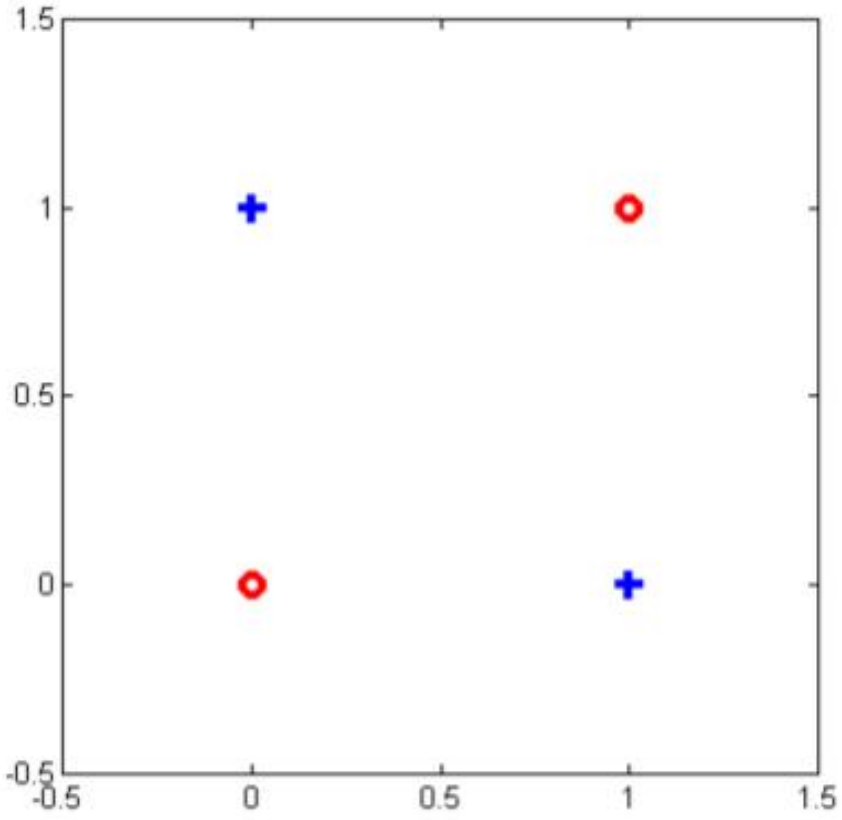

Figura 2.5: Función *XOR*, la clase 0 es representada con círculos, la clase 1 con cruces.

los datos. Lo que necesitamos entonces es agregar un perceptrón adicional a nuestro modelo, de esta forma contaremos con dos funciones lineales que nos permitirán separar satisfactoriamente los valores en la Figura 2.5 en sus respectivas clases. Este ultimo modelo obtenido es conocido como *perceptrón multicapa*, *red neuronal* o mas específicamente *redes feedforward*. A continuación revisaremos su arquitectura básica.

2.2.2 *Redes Feedforward*

Este tipo de algoritmo recibe el nombre de *"red"* porque se construye componiendo funciones (o perceptrones). Con respecto al termino *"feedforward"*, este proviene del inglés y hace referencia a que la información siempre fluye hacia adelante, desde una neurona hacia otra, donde no existen conexiones entre la salida y la entrada de una misma neurona. El modelo resultante se representa como un grafo acíclico mostrando como las funciones se conectan entre ellas y cómo fluye la información.

La arquitectura de este tipo de modelos está se puede dividir en tres partes principales, denominadas *capas*, tal como se muestra en

la Figura 2.6. Las capas en este tipo de redes se conocen como *completamente conectadas* (o *fully connected*) debido a que cada neurona de una capa se conecta con todas las neuronas en la capa siguiente pero nunca con neuronas de la misma capa.

En la Figura 2.6 podemos observar las distintas capas que componen una red neuronal. En primer lugar tenemos la *capa de entrada* donde, tal como su nombre lo indica, se define cuántos valores de entrada tomará nuestro modelo, estos serán luego enviados a la primer *capa oculta*. En la Figura 2.6 tenemos cinco capas ocultas, las neuronas en la primer capa oculta reciben los mismos cuatro valores de entrada, los procesan y luego pasan el resultado a los neuronas en la siguiente capa, estas últimas a su vez se encargan de procesar los datos recibidos pudiendo tomar decisiones en un nivel más complejo y abstracto que las neuronas de la primer capa. Finalmente el resultado de la última capa oculta es enviado a la *capa de salida*, las neuronas en esta capa comúnmente no poseen función de activación. Esto ocurre porque el resultado de esta capa se usa para representar las probabilidades de cada clase (en un problema de clasificación) o algún tipo de valor real (regresión).

Resulta interesante resaltar que una red neuronal con al menos una capa oculta es un *aproximador universal*. Puede ser demostrado según [6] que dada cualquier función continua $f^*(x)$ y algún $\varepsilon > 0$, existe una red neuronal $f(x)$ con una capa oculta tal que $\forall x, |f^*(x) - f(x)| < \varepsilon$. En conclusión, una red neuronal puede aproximar cualquier función continua.

Si bien la propiedad descripta anteriormente resulta fascinante es de cierta forma inservible en la practica, pues empíricamente las redes neuronales con varias capas ocultas funcionan mejor que aquellas con sólo una de estas. También ocurre que, si bien una red neuronal con 3 capas ocultas puede resultar mejor que una con sólo 2, agregar más capas a nuestro modelo no garantizaran que este mejore.

2.2.3 *Funciones de Activación*

Supongamos que queremos usar el perceptrón para resolver un problema de regresión en el cual queremos predecir algún valor $y \epsilon \mathbb{R}$, si sólo usáramos la función de activación vista anteriormente las salidas de esta neurona podría tener dos valores posibles: 0 o 1, lo cual no se ajustaría demasiado al tipo de problema que intentamos resolver. Es por este motivo que existen diferentes tipos de funciones de activación, si bien en la practica algunas funcionan mejor que otras, nuestra elección dependerá del tipo de datos y el problema con el que estamos trabajando o quizá, con la que hemos obtenido mejores resultados. A continuación mostramos brevemente las más populares.

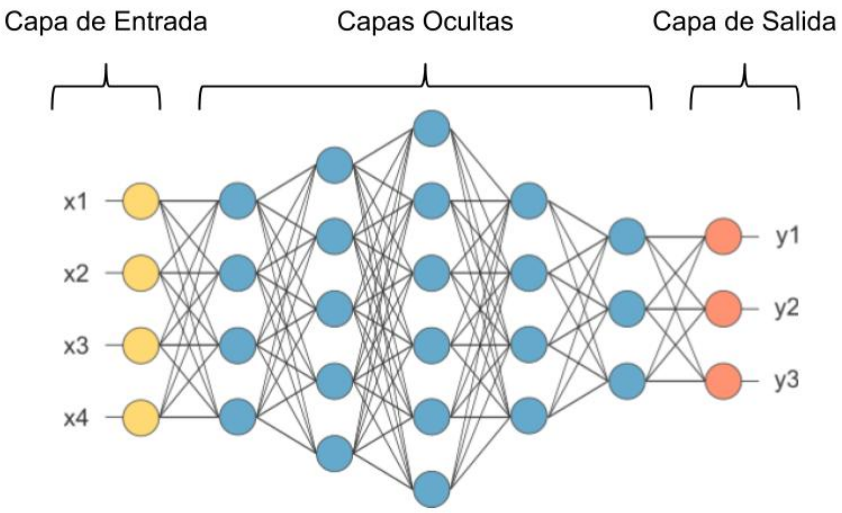

Figura 2.6: Arquitectura básica de una red neuronal.

- *Sigmoide*: Es definida mediante la expresión matemática

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

, se puede observar su forma en la Figura 2.7. Esta función toma números reales y los mapea dentro del rango entre cero y uno, aquellos números negativos grandes se convierten en cero mientras que los números positivos grandes se convierten en uno.

- *ReLU*: Esta computa la función $ReLU(x) = max(0, x)$, convirtiendo así todos los números negativos a cero y dejando los números positivos sin cambiar tal como se puede ver en la Figura 2.8.

- *Tanh*: Es una versión escalada de la función Sigmoide que se puede definir como $Tanh(x) = 2\sigma(2x) - 1$. Esta escala los números en el rango $[-1, 1]$ y está centrada en cero (Figura 2.9).

2.3 APRENDIZAJE VÍA MINIMIZACIÓN DEL ERROR EMPÍRICO

Supongamos que queremos predecir precios de casas basándonos en la cantidad de metros cuadrados de estas, este seria entonces un problema de regresión. Sean $X = \{x_1, x_2, ..., x_n\}, Y = \{y_1, y_2, ..., y_n\}$ para algún $n \in \mathbb{N}$, nuestros conjuntos con medidas de casas y sus respectivos precios. Queremos encontrar una función f, tal que $f(x_i; W, b) = y_i$. Luego necesitamos saber si nuestra función f ha logrado aproximar correctamente los datos, podemos hacer esto utilizando la siguiente fórmula conocida como error cuadrático medio:

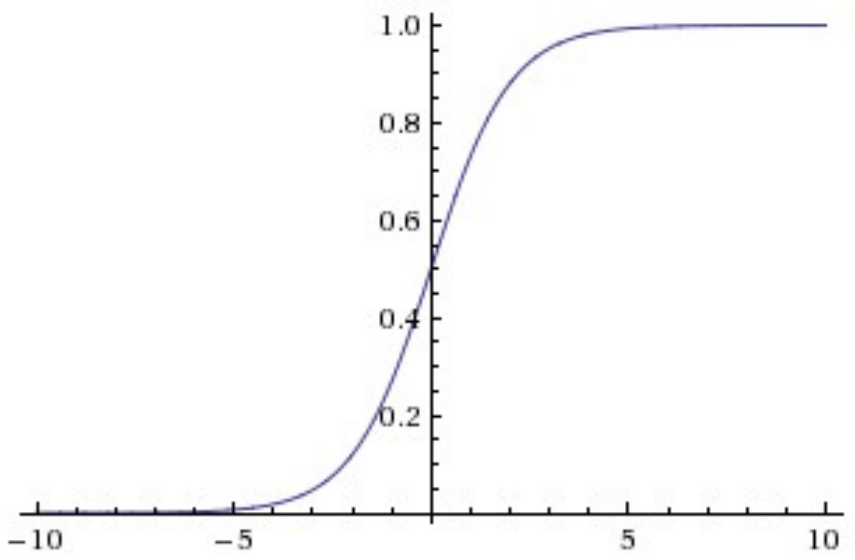

Figura 2.7: La función Sigmoide envía los números reales dentro del rango $[-1, 1]$.

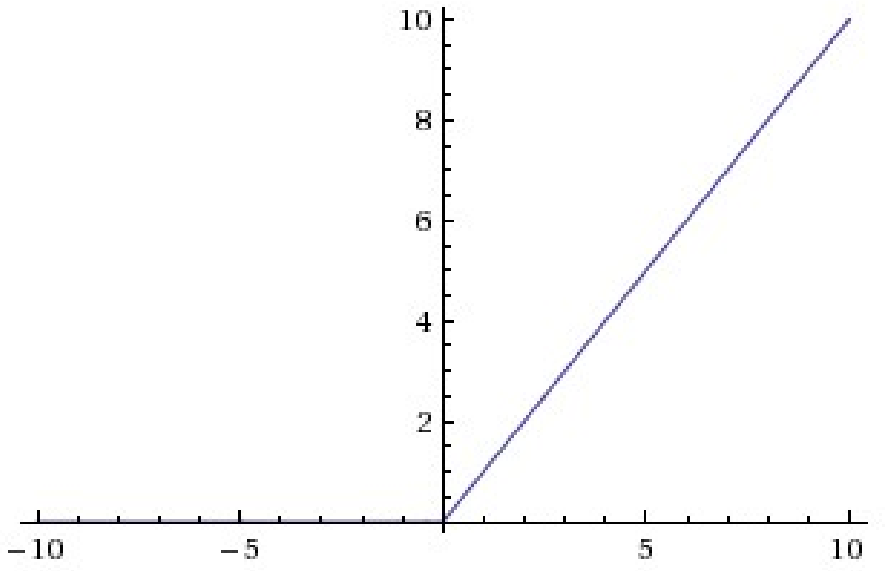

Figura 2.8: Función ReLU, es cero cuando $x < 0$ y luego lineal con pendiente 1 cuando $x > 0$.

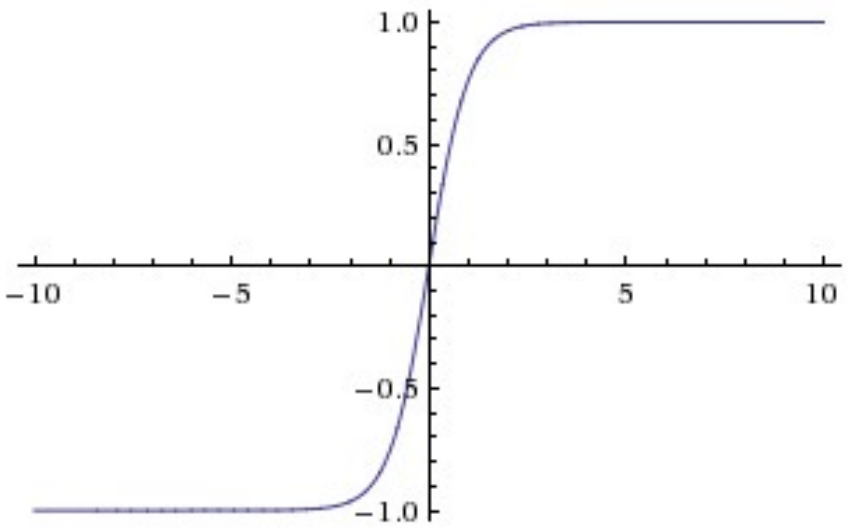

Figura 2.9: Tanh mapea los números en el rango $[-1, 1]$, centrada en cero.

$$L\left(X, Y; W, b\right) = \frac{1}{n}\sum_{i=1}^{n}\left(f\left(x_i; W, b\right) - y_i\right)^2 \tag{2.6}$$

L será la *función de costo* o también conocida como *función de pérdida*, está cuantificara que tan buenas son las predicciones hechas por nuestro modelo. Dado que L está computando el error de las predicciones, nuestro objetivo será minimizar esta función, es decir $argmin_{W,b}L\left(X, Y; W, b\right)$, y con esto minimizar el error en f. En la siguiente sección introduciremos un simple algoritmo para este fin conocido como *descenso del gradiente*.

2.4 APRENDIZAJE POR DESCENSO DEL GRADIENTE

Sea $L : \mathbb{R}^D \to \mathbb{R}$ una función de costo como la vista en la sección anterior y sea w un vector en $\mathbb{R}^D$ (los parámetros de nuestra función f descrita en la sección anterior, por simplicidad asumimos $w = [W, b]$). El gradiente de L en w, denotado como $\nabla L\left(w\right)$, es el vector de derivadas parciales de L, es decir, $\nabla L\left(w\right) = \left(\frac{\partial L(w)}{\partial w_1}, \frac{\partial L(w)}{\partial w_2}, ..., \frac{\partial L(w)}{\partial w_d}\right)$.

El descenso del gradiente es un algoritmo iterativo, empezamos con valor inicial de w (por ejemplo $w = [0, 0, ..., 0]$) y entonces en cada iteración damos un paso en la dirección negativa del gradiente en el punto actual. Esto es, $w^{(t+1)} = w^{(t)} - \eta\nabla L\left(w^{(t)}\right)$, donde $\eta > 0$, conocido como el coeficiente de aprendizaje, es el encargado de decidir el tamaño del paso que damos.

Intuitivamente, dado que el gradiente apunta a la dirección en que L crece alrededor de $w^{(t)}$, el algoritmo toma un pequeño paso en la dirección opuesta, decrementando el valor de la función L. Eventualmente, luego de T iteraciones obtendremos el vector final $w^{(T)}$para el

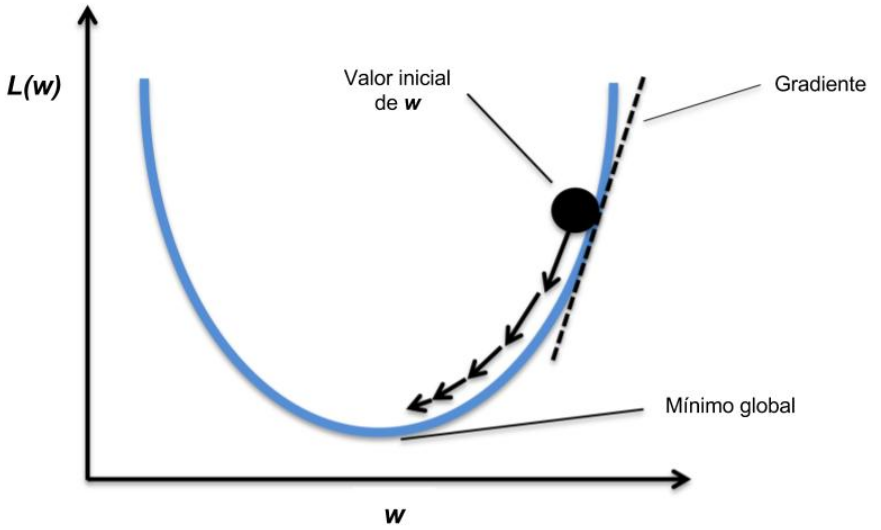

Figura 2.10: El descenso del gradiente iterativamente determina el valor de w que minimiza L.

cual, idealmente, $L\left(w^{(T)}\right)$ es un mínimo global. El algoritmo completo para este procedimiento puede observarse a continuación:

Algorithm 2.1 Descenso del gradiente.

$t = 0$
while $t < T$ **do**
  $i = 0$
  while $i < d$ {iterar por cada valor de w} **do**
    $w_i^{(t+1)} = w_i^{(t)} - \eta \nabla L\left(X, Y; w_i^{(t)}\right)$ {X, Y conjunto de entrenamiento}
    $i = i + 1$
  end while
  $t = t + 1$
end while

Finalmente, para retratar lo descrito con un ejemplo, asumamos $w \epsilon \mathbb{R}$ y sea $L : \mathbb{R} \to \mathbb{R}$ una función similar a la mostrada en la Figura 2.10, se comenzará con un valor de w inicial (cero o aleatorio) y en cada iteración de nuestro algoritmo, avanzando en la dirección opuesta del gradiente se irán dando pasos para obtener el valor óptimo de w, es decir, el mínimo valor posible en L.

2.5 REDES NEURONALES CONVOLUCIONALES

Las técnicas convencionales de aprendizaje automático están limitadas por su capacidad de procesar datos en su forma natural. Por décadas, construir un sistema inteligente para reconocimiento de pa-

trones requería un cuidadoso proceso de ingeniería y experiencia considerable acerca del área para diseñar un extractor de características que transforme los datos de entrada (por ejemplo los pixeles en una imagen) en una representación adecuada o vector de características a partir del cual algún algoritmo de aprendizaje automático nos permitiera detectar o clasificar patrones en los datos originales.

El aprendizaje por representación es un conjunto de métodos que permiten a la máquina ser alimentada con datos en su forma natural y automáticamente descubrir las representaciones necesarias para detección o clasificación. Los métodos de redes convolucionales se encuadran dentro de esta categoría, pero aplicando múltiples niveles de representación, obtenidos a través de la composición de módulos que transforman sucesivamente los datos (comenzando por los datos de entrada) en representaciones con un nivel más complejo y abstracto. El aspecto clave de esta arquitectura es que estas representaciones que obtenemos no están siendo diseñadas por ingenieros humanos, estas son inferidas a partir de los datos utilizando un algoritmo de aprendizaje genérico.

En las arquitecturas de redes neuronales introducidas anteriormentes vimos que cada neurona se conecta con todas las neuronas de la capa siguiente. Supongamos que queremos entrenar una red neuronal que tome imágenes como dato de entrada, este tipo de arquitectura no tiene en cuenta la espacialidad en la imagen dado que conecta todos los píxeles con una misma neurona y puede que la información en los pixeles de la esquina superior no tenga nada que ver con los pixeles de la esquina inferior. Otro problema con este tipo de redes es la gran cantidad de memoria que se necesita a medida que se agregan más capas, limitando así su capacidad de aprendizaje. Las redes convolucionales hacen frente a estos problemas usando tres ideas básicas: *campo receptivo local*, *pesos compartidos* y *agrupación*.

2.5.1 *Campo receptivo local*

En las redes neuronales vistas anteriormente las entradas eran descritas como un vector de neuronas, en las redes convolucionales es mejor pensar la entrada como una matriz de neuronas, así por ejemplo los valores de intensidad de una imagen de NxN píxeles se corresponden con una matriz NxN en la capa de entrada de la red. Como hacíamos anteriormente conectamos las neuronas de entrada a las neuronas de la capa oculta, pero esta vez no conectaremos cada dato de entrada con cada neurona oculta. Más precisamente, conectaremos cada neurona oculta con una pequeña región de nodos de entrada, por ejemplo, una región de $5x5$ en una imagen de entrada de $25x25$ píxeles tal como se muestra en la Figura 2.11.

La región seleccionada en la matriz de entrada es llamada el campo receptivo local de la neurona oculta. Esta neurona aprende un valor

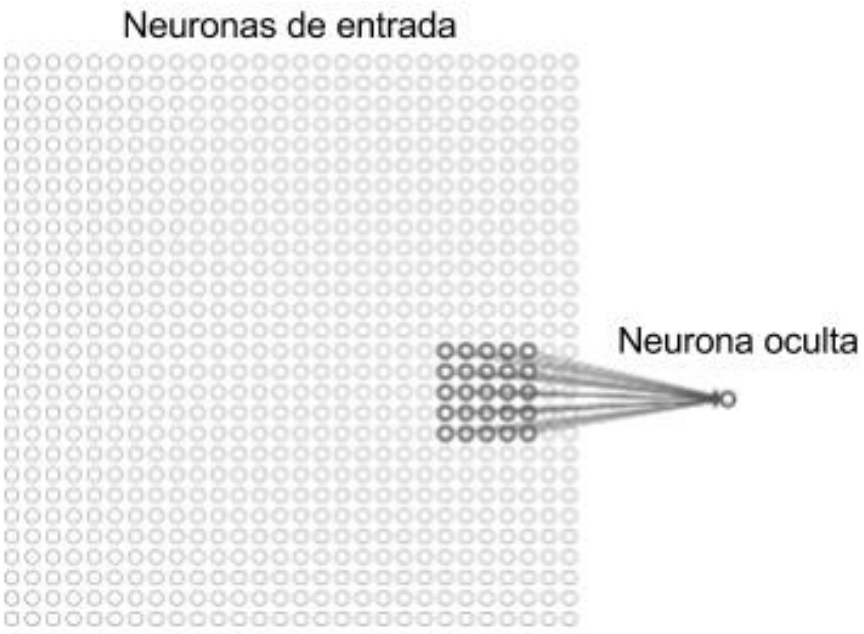

Figura 2.11: Campo receptivo local para una neurona oculta.

de peso por cada dato en el campo receptivo y un bias. De esta manera cada neurona oculta está aprendiendo a representar pequeñas regiones en la imagen. Por cada campo receptivo local tenemos una neurona oculta a la cual este está conectado, para esto vamos deslizando el campo receptivo horizontal y verticalmente a través de la matriz con valores de entrada tal como se muestra en la Figura 2.12. En este ejemplo tenemos una matriz de entrada de $28x28$ neuronas y un campo receptivo local de $5x5$ el cual desplazamos de a una neurona, dando como resultado una capa oculta con $23x23$ neuronas. La cantidad de neuronas que "saltamos" al desplazar el campo receptivo local se conoce como *paso* (o *stride* por su nombre en inglés), aquí usamos un stride de valor 1 pero se pueden usar otros, lo que se hace generalmente es probar distintos valores y utilizar el que funcione mejor.

2.5.2 *Pesos compartidos*

Ya vimos que cada neurona oculta tiene un valor de bias y una matriz de pesos de tamaño NxN, o mas precisamente $5x5$ en el ejemplo anterior, correspondiente a su campo receptivo local, lo que no mencionamos es que se usarán los mismos pesos y bias para cada una de las $24x24$ neuronas en la capa oculta. Más formalmente, para la (j,k)-ésima neurona oculta, su valor será:

$$f\left(b + \sum_{l=0}^{4}\left(\sum_{m=0}^{4}\left(w_{l,m}a_{j+l,k+m}\right)\right)\right). \tag{2.7}$$

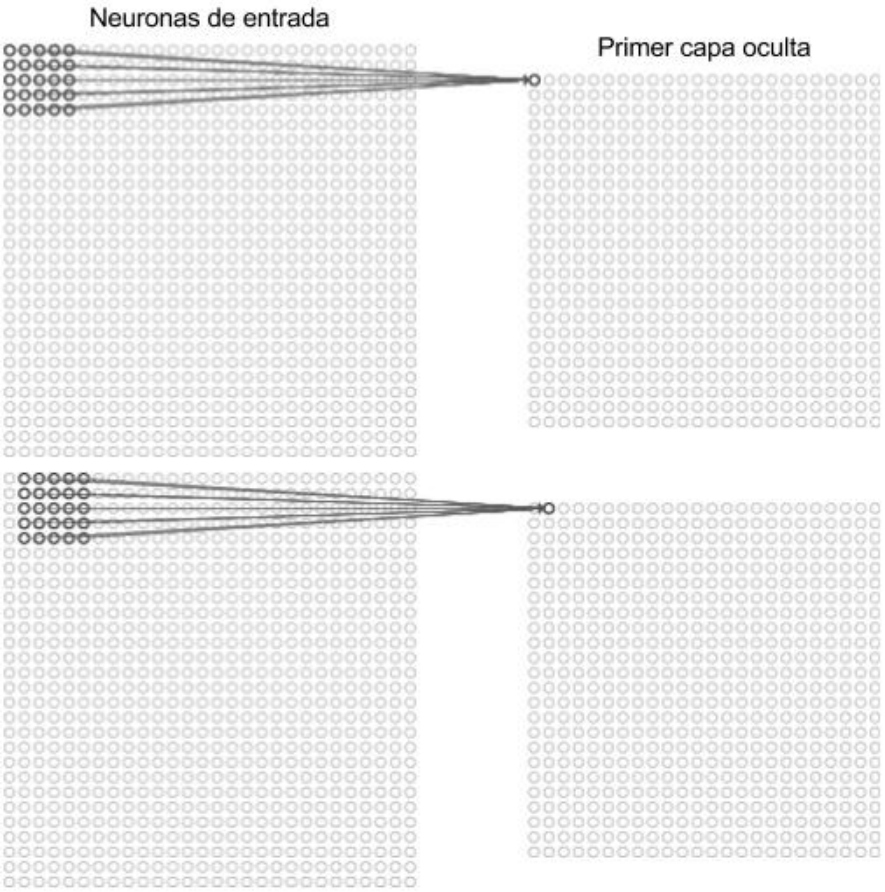

Figura 2.12: Desplazamiento del campo receptivo local.

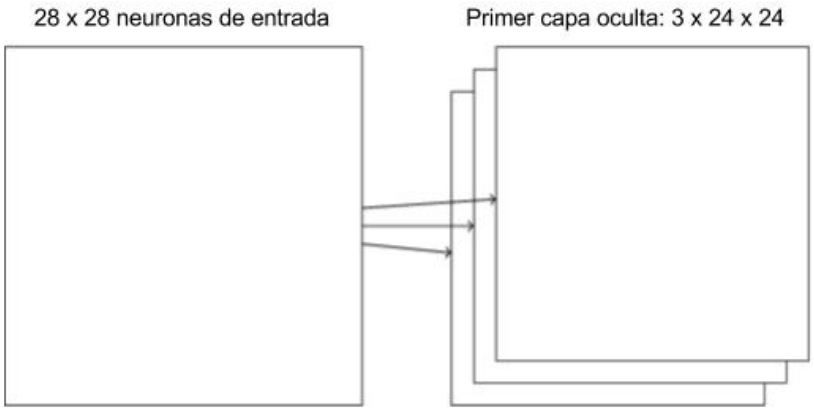

Figura 2.13: Resultado de una convolución con 3 kernels.

Donde f es la función de activación, b es el valor compartido de bias, w es una matriz $5x5$ con los pesos compartidos y finalmente $a_{x,y}$ es el valor de entrada en la posición (x, y). w es comúnmente llamada *kernel* y al ser compartido significa que todas las neuronas de la primer capa oculta detectan el mismo patrón, pero en diferentes lugares de la imagen de entrada. Esto es sumamente útil y hace este tipo de redes robustas a invariancias en la posición de los objetos.

En el ejemplo anterior solo tenemos un kernel y esto significa que la primer capa oculta sólo puede detectar un único tipo de patrón, por este motivo es que agregaremos mas kernels, supongamos por ejemplo que tenemos 3 kernels, obteniendo entonces un modelo como el de la Figura 2.13, como se puede ver, dentro de la capa oculta tenemos a su vez 3 capas, cada una correspondiente a un kernel diferente. Este tipo de capa oculta es conocida como *capa convolucional*, su nombre proviene de la fórmula de activación vista anteriormente a la cual se la llama comúnmente *convolución*.

2.5.3 *Agrupación*

Además de la capa convolucional, este tipo de redes también poseen una capa llamada *agrupación* (o *pooling* por su nombre en inglés). La capa de agrupación se suele utilizar inmediatamente después de la capa convolucional, su función es reducir y simplificar la información de salida de la capa de convolución. Cada unidad en la capa de agrupación contiene información condensada de una región KxK de neuronas de la capa anterior. Existen varios tipo de capas de agrupación pero la más usada se conoce como *max-pooling*, en esta, cada unidad en la capa de agrupación retorna el valor más grande de la región de entrada. Un ejemplo se puede observar en la Figura 2.14, donde tenemos una capa oculta de $24x24$ neuronas (resultado de una capa convolucional), a esta le aplicamos una capa de agrupación con

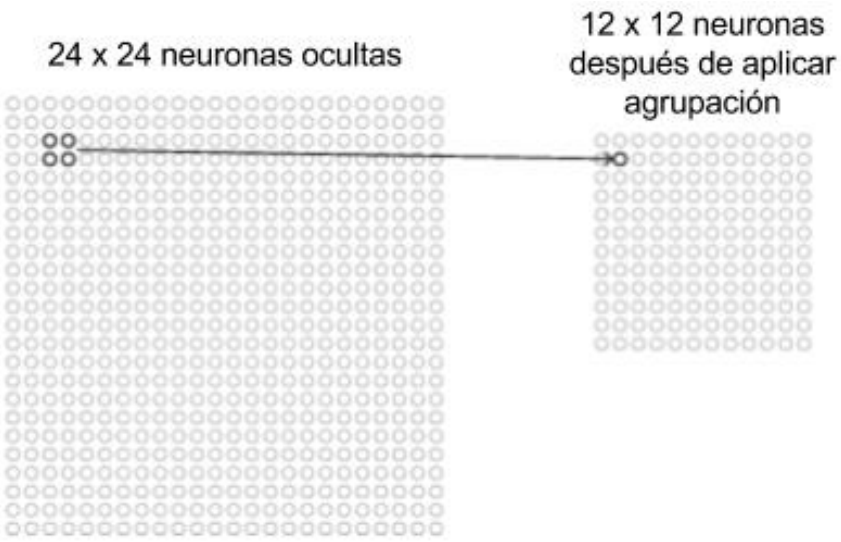

Figura 2.14: Resultado obtenido al aplicar agrupación.

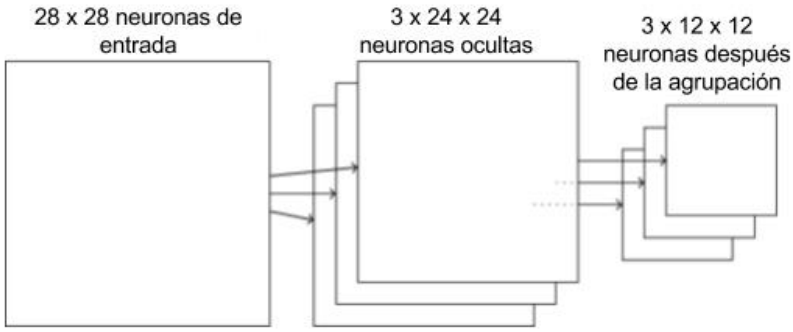

Figura 2.15: Agrupación después de una convolucion con 3 kernels.

una región de $2x2$ obteniendo como resultado una salida de $12x12$ neuronas.

Como vimos anteriormente, las capas convolucionales tienen usualmente más de un kernel, por lo tanto al aplicar la capa de agrupación deberemos hacerlo por separado para la salida de cada kernel en la capa de convolución. Por ejemplo, si queremos aplicar agrupación y tenemos 3 kernels nos quedaría una estructura como similar a la mostrada en la Figura 2.15.

2.5.4 *Ejemplo: clasificación de imágenes con redes convolucionales*

Ahora que hemos visto los conceptos básicos que forman parte de la estructura de una red convolucional podemos mostrar la arquitectura completa de este tipo de redes con el siguiente ejemplo:

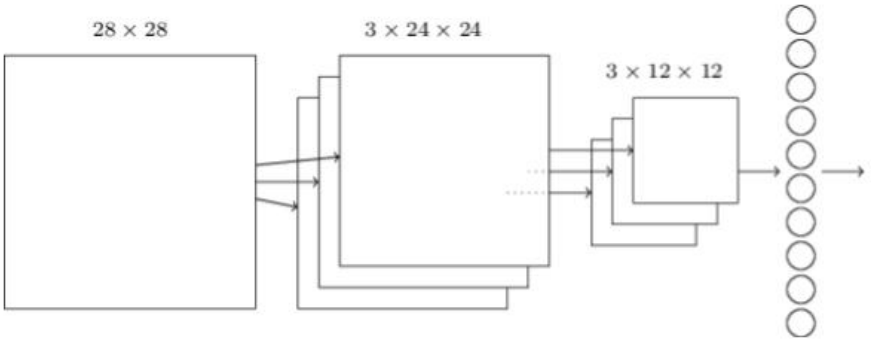

Figura 2.16: Arquitectura de red convolucional para clasificación entre 10 categorías.

Supongamos que queremos clasificar imágenes entre 10 categorías posibles, para esto usaremos una red convolucional como la descripta en la Figura 2.16, esta red toma como entrada una matriz de 28x28 elementos correspondientes a los valores de una imagen de 28x28 píxeles, usamos una capa convolucional con 3 kernels de tamaño 5x5 y un stride de 1. Luego tenemos una capa de agrupación que se aplica sobre los valores de salida de cada kernel en la capa de convolución. La última capa que agregaremos al modelo tendrá 10 neuronas (una por cada clase que queremos predecir), cada neurona en esta capa se conectara con todas las neuronas de la capa de agrupación, algo similar a lo que hacíamos en las redes neuronales completamente conectadas vistas al principio.

3

RECONOCIMIENTO FACIAL

Ahora que hemos visto los conceptos básicos acerca de algoritmos de aprendizaje automático y, en particular, sobre redes neuronales convolucionales, podemos empezar a explorar los aspectos técnicos en el desarrollo de sistemas de reconocimiento facial. Comenzaremos analizando las dos problemáticas principales dentro de reconocimiento, conocidas como *verificación* e *identificación*, luego estudiaremos cada una de las cuatro etapas generales que componen estos sistemas (detección, normalización, extracción de características y reconocimiento) dando detalles sobre su arquitectura y los resultados alcanzados en la actualidad.

3.1 VERIFICACIÓN E IDENTIFICACIÓN FACIAL

El problema de reconocimiento facial se clasifica principalmente en dos modelos:

- Verificación facial
- Identificación facial

En *verificación facial* se realiza una comparación uno a uno entre dos rostros para entonces confirmar si corresponden a la misma identidad o no. En este tipo de modelos se tiene almacenado un perfil biométrico (en este caso particular, una imagen facial) correspondiente a alguna identidad conocida, luego, al recibir un nuevo dato biométrico se procede a comprar esta con el perfil almacenado y se intenta comprobar si el individuo es quien dice ser. Un ejemplo actual de este tipo de sistemas son los teléfonos móviles modernos que permiten ser desbloqueados por su propietario mediante una fotografiá tomada en el momento.

Por otro lado, cuando hablamos de *identificación facial*, se trata de una comparación uno a varios, entre un rostro cuya identidad desconocemos contra una base de datos de rostros conocidos para determinar la identidad de la nueva imagen. Este tipo de modelos pueden ser encontrados por ejemplo en Facebook, cuando una persona carga una fotografía, el programa detecta los rostros presentes en la misma para luego intentar identificarlos basándose en los contactos presentes en el perfil del usuario y así ofrecer recomendaciones acerca de quienes aparecen en la imagen para ser "etiquetados".

Para ser capaces de ejecutar alguno de los modelos descritos se necesita primero un algoritmo que nos permita transformar la imagen

del rostro de una persona en una representación matemática de este, por ejemplo, vectores de características tales que la distancia entre dos vectores de la misma persona sea pequeña (idealmente cero) y la distancia entre vectores de diferentes personas sea grande (idealmente el máximo valor posible). Para esto utilizaremos algoritmos de aprendizaje automático como los que explicaremos mas adelante.

3.2 ETAPAS EN EL RECONOCIMIENTO FACIAL

La tarea de reconocimiento facial se divide en cuatro etapas principales:

- Detección facial.
- Normalización y alineamiento.
- Extracción de características.
- Reconocimiento.

Dada cierta imagen, nuestro primer paso será identificar si esta contiene algún rostro y las coordenadas de este. Para tal tarea será necesario contar con un algoritmo de *detección facial*, uno de los más populares fue el introducido por Viola y Jones en 2001 [34] gracias a su velocidad, permitiendo así su uso en dispositivos con hardware poco potente tal como cámaras fotográficas económicas, otra de sus ventajas fue su alto nivel de precisión, con un bajo margen de detecciones en regiones donde no había un rostro presente. En 2005 surgen los *Histogramas de Gradientes Orientados* (o *HOG* por las siglas de su nombre en inglés) [7] obteniendo una mejora sustancial en comparación con el método anterior, principalmente logrando detectar un mayor numero de rostros que no eran captados por el algoritmo anterior; sin embargo, en la actualidad son las redes neuronales convolucionales quienes resultaron en el método más eficaz para esta tarea [36].

Una vez obtenida la posición del rostro en la imagen, este es recortado para ser *normalizado* en términos de tamaño, color, iluminación, entre otros; así como también *alineado* para llevar este a una posición canónica que sea similar a la de otros rostros con los cuales se comparará en la etapa de reconocimiento. Para el proceso de alineamiento se consideran como puntos de referencia partes específicas de la cara como ojos, nariz, labios, etc.

Luego de esto, necesitamos transformar la imagen obtenida en una representación matemática de esta. Comúnmente conocida como *vector de características*, esta contendrá información descriptiva sobre el rostro de forma tal que si contamos con una función que compare este tipo de vectores, podremos saber si dos de estos corresponden a la misma persona, o si un vector dado pertenece a la identidad de alguna persona en una base de datos con vectores de este estilo (*reconocimiento*).

Una descripción exhaustiva sobre las dos últimas etapas aquí descritas será aborda en las próximas secciones.

3.3 REDES CONVOLUCIONALES EN RECONOCIMIENTO FACIAL

El primer sistema semiautomático para reconocimiento facial fue desarrollado en los 60 y requería de un humano para señalar puntos de interés en la imagen tales como ojos, nariz, labios y oídos, a partir de estos se computaban métricas de radio y distancia en base de puntos de referencia genéricos para luego comparar estas con una base de datos de referencia. Más tarde en los 70, Goldstein, Armor y Lesk utilizaron 21 características subjetivas del rostro como el color del cabello y espesor de los labios. El problema con ambos métodos es que las características descriptivas de un rostro y su posición debían ser computadas manualmente, conllevando así, un complicado proceso de ingeniería y alto conocimiento sobre el tema, un paradigma que cambió con la llegada de las redes neuronales convolucionales.

En 2012 las redes neuronales convolucionales pasaron a un primer plano obteniendo el primer puesto en la competencia ImageNet y superando por una gran diferencia en precisión a otros métodos que hasta el momento eran considerados el estado del arte en problemas de aprendizaje automático. Cuando se emplean redes convolucionales en la tarea de identificación facial, el procedimiento general es utilizar estos modelos como clasificadores, siendo entrenados en bases de datos con millones de imágenes correspondientes a miles de identidades distintas, estas ultimas serán las clases a predecir luego por el modelo obtenido. En la tarea de verificación facial, se suele llevar a cabo el mismo proceso de entrenamiento descripto para identificación pero una vez finalizado este, se quita la capa de clasificación de la red y el modelo resultante se utiliza como generador de vectores de características a partir de imágenes, obteniendo estos desde la salida de una capa convolucional o una capa completamente conectada; el objetivo del entrenamiento previo fue adaptar el modelo a un dominio especifico de datos (en este caso, imágenes de rostros). Una vez obtenidos los vectores de características a partir de las imágenes, y utilizando alguna función para medir distancias entre vectores, podemos proceder a realizar la comparación uno a uno requerida en verificación facial.

La primer red convolucional exitosa para verificación facial surge en el año 2014 de la mano del equipo de investigación de Facebook, llamada DeepFace [31], esta estaba compuesta por 8 capas y entrenada sobre una extensa base de datos de 4 millones de imágenes correspondientes a 4000 personas. Otra contribución de [31] es el alineamiento 3D de rostros para llevar todas imágenes a una misma posición entre ellas, este tipo de alineamiento requiere de un costoso proceso de cómputo y, en trabajos futuros [29] [27], se demostró

que este alineamiento no era crítico para lograr mejores resultados y que podía ser reemplazado sin problemas por un alineamiento 2D, fácil de computar a través de una transformación afín. En 2015 Google introduce FaceNet [24], mostrando asombrosos resultados con la utilización de una nueva función de costo conocida como *Pérdida de Triplete* (o *Triplet Loss* por su denominación en inglés) aplicada a la arquitectura de red convolucional Inception [30] y un conjunto de datos de entrenamiento de 200 millones de imágenes de 8 millones de identidades distintas. Un poco más adelante en ese año el grupo de investigación de la universidad de Oxford propone usar la misma función de costo de FaceNet pero entrenando un modelo con un conjunto de datos menor a lo que comúnmente se utilizaba (2,6 millones de imágenes de 2622 personas), logrando igualmente resultados cercanos al estado del arte en verificación facial pero con un entrenamiento más accesible. En la sección de experimentos podrá observar una tabla comparando los modelos descritos y sus resultados. En el próximo capítulo veremos en detalle la función de Triplet Loss, sus ventajas y desventajas, y las distintas variantes de esta que han sido propuestas en los últimos años.

3.4 RECONOCIMIENTO FACIAL CON TRIPLET LOSS

Una de las claves en el éxito de las redes convolucionales es la posibilidad de contar con un amplio conjunto de datos para entrenamiento. Lamentablemente, en el mundo del reconocimiento facial la posibilidad de acceder a extensas bases de datos faciales es escasa, restringiendo los avances en este área a grandes empresas como Google, Facebook, Baidu, etc. Por ejemplo el sistema más reciente de reconocimiento facial de Google [24] fue entrenado usando 200 millones de imagenes y 8 millones de identidades diferentes. En esta sección y la siguiente veremos cómo, a pesar de contar con una base de datos varios órdenes de magnitud menor a la mencionada, se puede lograr entrenar un sistema de reconocimiento facial que reporte resultados cercanos al estado del arte.

Existe una serie de pasos comunes en la construcción de tales sistemas, primero se comienza entrenando una red neuronal para identificación facial, es decir, entrenada para clasificar la identidad de un rostro entre un conjunto finito de identidades posibles, luego se quita la capa de clasificación de dicha red y el modelo resultante es usado para convertir imágenes en vectores de características. A partir de estos, se utilizan métodos como PCA [37], Joint Bayesian [29] [27] o metric-learning [24], para aprender nuevas representaciones de menor dimensionalidad y con mejor desempeño en la tarea de verificación facial. En nuestros experimentos utilizaremos la red neuronal VGG-Face [21] para generar una representación vectorial de las imá-

Figura 3.1: Triplet Loss minimiza la distancia entre un ancla y un positivo, los cuales tienen la misma identidad; mientras que maximiza la distancia entre el ancla y un ejemplo negativo (diferente identidad).

genes en primera instancia, estas luego se utilizaran para aprender nuestro algoritmo de Triplet Loss detallado a continuación.

Queremos generar una representación $f(x)$, de una imagen facial x en un vector de características en $\mathbb{R}^L$, $L \epsilon \mathbb{N}$, tal que la distancia entre todos los rostros, independientemente de las condiciones de la imagen, de la misma identidad sea pequeña, mientras que la distancia de un par cualquiera de imágenes de distinta identidad sea grande. Formalmente, buscamos asegurarnos que una imagen x_a (ancla) de una persona específica, sea más cercana a las demás imágenes x_p(positiva) de la misma persona que a cualquier imagen x_n (negativa) de otra persona, Figura 3.1. Sea entonces $DE(x, y) : \mathbb{R}^L x \mathbb{R}^L \rightarrow \mathbb{R}$ una función que mida la distancia entre dos vectores obtenidos a partir de f, queremos que se cumpla,

$$DE\left(f\left(x_a\right), f\left(x_p\right)\right) + \alpha < DE\left(f\left(x_a\right), f\left(x_n\right)\right), \forall \left(x_a, x_p, x_n\right) \epsilon T \quad (3.1)$$

donde α es un margen impuesto entre los pares positivos y negativos, y T es el conjunto de triplets usados en el entrenamiento.

En todos nuestros experimentos los vectores x_a, x_p y x_n serán puntos en un espacio euclídeo n-dimensional, así también como $f(x_a)$, $f\left(x_p\right)$ y $f(x_n)$. Finalmente DE sera la distancia euclidiana al cuadrado que es calculada de la siguiente manera: para $X = (x_1, x_2, ..., x_n)$ e $Y = (y_1, y_2, ..., y_n)$,

$$DE(X, Y) = (x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2 \quad (3.2)$$

3.4.1 *Entrenamiento*

Dada una imagen x, definiremos con $\theta(x) \epsilon \mathbb{R}^D, D \epsilon \mathbb{N}$, la representación obtenida a partir de una red neuronal, esta es normalizada utilizando la norma euclidiana y luego proyectada a un espacio L-dimensional ($L << D$) usando una proyección afín tal que $f(x) = W \frac{\theta(x)}{\|\theta(x)\|_2}$. Esta matriz de proyección $W \epsilon \mathbb{R}^{LxD}$ es entrenada para minimizar el Tripet Loss empírico

$$E(W) = \sum_{(x_a, x_p, x_n)\epsilon T} max\left(0, \alpha + \psi\left(x_a, x_p, x_n\right)\right) \tag{3.3}$$

con

$$\psi\left(x_a, x_p, x_n\right) = DE\left(f\left(x_a\right), f\left(x_p\right)\right) - DE\left(f\left(x_a\right), f\left(x_n\right)\right) \tag{3.4}$$

Usaremos técnicas de descenso del gradiente para estimar la matriz de proyección, entonces necesitamos computar la derivada de nuestra función de costo tal como se muestra a continuación,

$$\nabla_W E = \begin{cases} 0 & \alpha + \psi\left(x_a, x_p, x_n\right) < 0 \\ \nabla_W \psi\left(x_a, x_p, x_n\right) & c.c. \end{cases} \tag{3.5}$$

luego, sea

$$\varsigma\left(x\right) = \frac{\theta\left(x\right)}{\| \theta\left(x\right) \|_2} \tag{3.6}$$

tenemos que,

$$\nabla_W DE\left(f\left(x_a\right), f\left(x_p\right)\right) = \nabla_W DE\left(W\varsigma\left(x_a\right), W\varsigma\left(x_p\right)\right) \tag{3.7}$$

$$= 2W\left(\varsigma\left(x_a\right) - \varsigma\left(x_p\right)\right)\left(\varsigma\left(x_a\right) - \varsigma\left(x_p\right)\right)^T \tag{3.8}$$

Análogo para el caso $\nabla_W DE\left(f\left(x_a\right), f\left(x_n\right)\right)$.

3.4.2 *Selección de triplets*

Generar todos los triplets posibles puede resultar en demasiados ejemplos que son fácilmente satisfechos (es decir, cumplen la restricción dada en Ec. 3.1), estos datos no van a contribuir al entrenamiento y producirán una convergencia más lenta. Por otro lado seleccionar los triplets que mejor violen la Ec. 3.1 puede llevarnos a converger a mínimos locales no óptimos durante las primeras etapas del entrenamiento. El método usado para buscar triplets se basa en extender cada par $\left(f\left(x_a\right), f\left(x_p\right)\right)$ a un triplet $\left(f\left(x_a\right), f\left(x_p\right), f\left(x_n\right)\right)$ eligiendo $f\left(x_n\right)$ aleatoriamente, pero sólo entre aquellas que cumplan,

$$DE\left(f\left(x_a\right), f\left(x_p\right)\right) + \alpha > DE\left(f\left(x_a\right), f\left(x_n\right)\right) \tag{3.9}$$

Esta es una forma útil de encontrar triplets informativos pero menos agresiva y más barata. Utilizando esta técnica y considerando un conjunto de entrenamiento con n clases y k muestras por clase, obtendríamos finalmente un total de $nk(k-1)(n-1)$ triplets.

4

EXPERIMENTOS

Utilizaremos la base de datos conocida como *"Labeled Faces in the Wild"* (*LFW*) [10] para todos nuestros experimentos dado que es la opción por defecto cuando se trata de medir y comparar la precisión de algoritmos para verificación facial. Este contiene 13233 imágenes faciales con 5749 identidades diferentes y, de estas, 1680 aparecen en dos o más imágenes.

En primer lugar, y como se explicó anteriormente, usaremos una red neuronal convolucional para convertir cada imagen de LFW a un vector de características que la represente. El modelo elegido para esta tarea es conocido como *VGG-Face* [19], el cual ha sido liberado por sus autores para usos no comerciales. En la Figura 4.1 se puede ver una descripción completa sobre la arquitectura *VGG16* en la cual se basa el modelo mencionado anteriormente. VGG-Face ha sido entrenado utilizando un conjunto de datos de 2,6 millones de rostros correspondientes a 2622 identidades, ninguna de estas perteneciente a la base de datos LFW. Inicialmente la red neuronal es entrenada para el problema de reconocimiento facial entre $N = 2622$ identidades distintas, utilizando para tal fin una función de *SoftMax* en la capa de salida. Dada una imagen de entrada a la red neuronal esta función retornará para cada una de las N identidades un valor de probabilidad utilizando la siguiente fórmula:

$$a_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}, \tag{4.1}$$

$1 \leq i \leq N, z_j$ peso en la neurona j.

Luego que nuestra red ha sido entrenada, la capa de SoftMax se quita y se utiliza como salida la última capa fully connected, denominada *"fc7"*, ahora por cada imagen de entrada a la red se obtendrá un vector de características de 4096 dimensiones como salida.

Con el modelo obtenido, y con el objetivo de obtener representaciones vectoriales robustas para cada imagen en LFW seguiremos el procedimiento descrito en [19], el cual consta de los siguientes pasos: dada la fotografía de un rostro, se generan cinco imágenes nuevas recortando una región de $224x224$ píxeles de cada esquina y el centro en la imagen original, esta última es rotada horizontalmente y el proceso se repite nuevamente (obteniendo en total 10 imágenes nuevas). Para permitir un análisis en múltiples escalas, el rostro es llevado a tres tamaños diferentes, 256, 384 y 512 píxeles; luego el proceso explicado anteriormente es ejecutado para cada uno de estos tamaños, obteniendo como resultado 30 imágenes considerando distintas

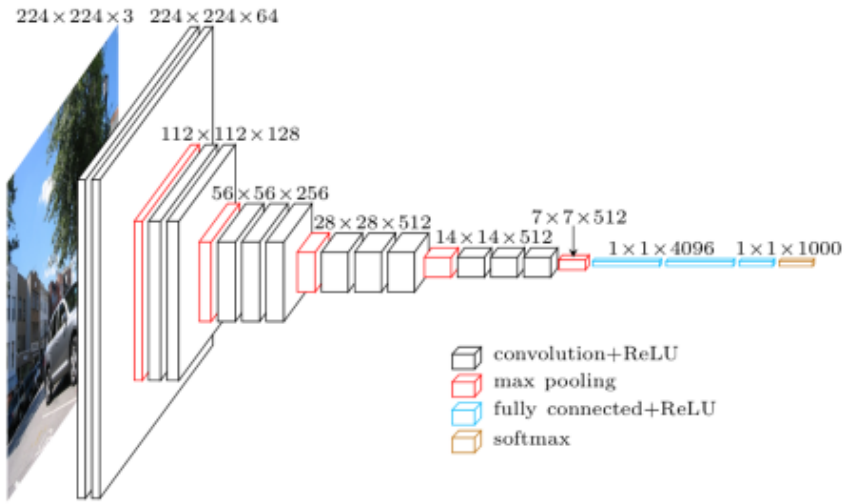

Figura 4.1: La arquitectura de red convolucional *VGG16* introducida por Simonyan y Zisserman en su publicación ([26]) de 2014.

regiones, estas serán enviadas como entrada a nuestra red neuronal obteniendo así de cada rostro treinta vectores de características de 4096 dimensiones, estos son entonces promediados resultando en un único vector con la misma dimensión que los anteriores.

Para reportar resultados en verificación, el conjunto de datos mencionado anteriormente provee una lista con 6000 pares de rostros, la mitad de estos correspondientes a la misma persona mientras que el resto corresponden a diferentes. Nuestra tarea será tomar cada uno y predecir si se trata del mismo individuo o no, esto se llevará a cabo comparando la distancia entre los vectores de características generados por cada rostro en el par y aceptándolos como la misma persona si la distancia es menor a un determinado margen τ, este último no es proveído y debe ser estimado con validación cruzada sobre el conjunto de pares.

Utilizaremos dos métricas para medir nuestro modelo, la primera y más comúnmente usada se denomina *Exactitud* (o *accuracy* en inglés) y está dada por la siguiente fórmula,

$$Exactitud = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.2}$$

, en la cual, TP se refiere a la cantidad de pares predecidos correctamente como la misma persona, TN es la cantidad de pares predecidos correctamente como distintas personas, FP es la cantidad de pares predecidos erróneamente como la misma persona y, finalmente, FN contabiliza los pares predecidos erróneamente como distintas personas. La segunda métrica es conocida como *Tasa de Error Igual* (*EER* por sus siglas en inglés), esta es usada por [19], y dado que buscamos replicar sus pasos, es necesaria para poder comparar los resultados obtenidos en nuestros experimentos. Esta métrica es definida como

la tasa de error en el punto de la curva ROC donde la cantidad de falsos positivos y falsos negativos son iguales, cuanto más bajo es el EER se considera que el sistema es más exacto. La ventaja de esta con respecto a Exactitud es que su resultado es independiente del margen τ.

Durante la etapa de extracción de vectores de características a partir de imágenes, en [19] utilizan un detector de rostros para ajustar la zona facial en la imagen y eliminar cualquier interferencia causada por elementos en segundo plano, además de este también se emplea un algoritmo de alineamiento 2-dimensional, el cual no es provisto, para llevar todos los rostros a una posición canónica. En nuestros experimentos se intentó reproducir este preprocesamiento pero no se obtuvieron mejoras, en cambio, nuestro mejor resultado se obtuvo utilizando las fotos de LFW sin modificar, esto puede deberse en gran medida por emplear un algoritmo de alineamiento inadecuado. Sin embargo, si observamos el Cuadro 4.2 comparando ambos experimentos, se puede ver que la ganancia obtenida con métodos de detección de rostros y alineamiento es baja, permitiéndonos así contar igualmente con un buen sistema para verificación facial que no requiera de ellos y, por ende, que requiera menos computo.

Para entrenar el algoritmo basado en Triplet Loss que introdujimos en la sección anterior realizamos una implementación del mismo en el lenguaje de programación Python [22], haciendo uso principalmente de la librería Numpy [35]. Utilizaremos como datos para el entrenamiento los vectores de características generados previamente sobre las imágenes de LFW.

En una primera instancia necesitamos estimar los parámetros de nuestro modelo que nos permitan obtener el mejor resultado, para esta tarea LFW provee dos conjuntos de pares denominados *pairsDevTrain* (para entrenar nuestro modelo) y *pairsDevTest* (para medir la exactitud), cabe destacar que las identidades entre ambos conjuntos son diferentes. Durante el entrenamiento, utilizaremos los vectores de características correspondientes a las personas listadas en pairsDevTrain. Una vez aprendido nuestro modelo, este es utilizado para medir su exactitud en la tarea de verificación facial sobre el conjunto pairsDevTest. El mejor resultado obtenido a través de estos experimentos fue utilizando como método de optimización el algoritmo Adam [11], con una tasa de aprendizaje de 0,1 durante diez iteraciones; cada iteración se realizó sobre todo el conjunto de entrenamiento.

Dados los parámetros estimados previamente, procedemos a entrenar y validar nuestro modelo definitivo siguiendo las directrices para reportar resultados en LFW, para tal fin tomaremos el conjunto de 6000 pares de identidades utilizado para medir algoritmos en verificación facial, estos pares se encuentran distribuidos en 10 conjuntos del mismo tamaño, donde ningún conjunto posee una identidad ya contenida en otro. Entrenaremos el algoritmo de Triplet Loss

Experimento	Detección de rostro	Alineamiento	$100\,\% - \text{EER}$
VGG-Face (sin Triplet Loss) [19]	Si	Si	97,27
Nuestro	No	No	97,1

Cuadro 4.2: Comparativa entre experimentos, en ambos casos se utilizó la distancia euclidiana al cuadrado para comparar los vectores de características.

con validación cruzada en 10 iteraciones (10-*fold cross-validation* por su nombre en inglés), utilizando en cada iteración los vectores de características de las personas contenidas en 9 conjuntos de pares para entrenamiento, mientras que, los demás pertenecientes al conjunto restante, se utilizaran para la evaluación del modelo entrenado. El resultado final estará dado por el promedio de los resultados en cada iteración. Finalmente en el Cuadro 4.3 tenemos una comparativa de nuestro método con otros mencionados anteriormente.

Modelo	Imágenes	Identidades	Exactitud %	$100\,\% - \text{EER}$
DeepFace [31]	$4M$	$4K$	97,35	-
Fusion [32]	$500M$	$10M$	98,37	-
FaceNet [24]	$200M$	$8M$	99,63	-
VGG-Face (con Triplet Loss) [19]	$2{,}6M$	$2{,}6K$	98,95	99,13
Nuestro (con Triplet Loss)	$2{,}6M$	$2{,}6K$	$97{,}91 \pm 00{,}61$	$97{,}71 \pm 00{,}61$

Cuadro 4.3: Comparativa de exactitud entre modelos evaluados en LFW considerando la cantidad de datos utilizados durante su entrenamiento.

5 CONCLUSIONES Y TRABAJO FUTURO

El trabajo de esta tesis se enfocó principalmente en la investigación del problema de reconocimiento facial con redes neuronales convolucionales y, en la utilización complementaria del algoritmo de Triplet Loss con el objetivo de obtener modelos de reconocimiento facial que alcancen resultados similares al estado del arte pero, requiriendo para su entrenamiento un conjunto de datos varios ordenes de magnitud menor comparado a aquellos utilizados comúnmente.

RESUMEN GENERAL

Se comenzó introduciendo el problema de reconocimiento facial con aprendizaje automático y explicando a grandes rasgos las principales etapas que componen a este tipo de algoritmos. También se hizo una revisión de trabajos previos en el área.

Como es debido, se presentó la teoría acerca de aprendizaje automático, sus principales ramas y su aplicación en imágenes. Luego se puntualizo en el estudio de redes neuronales y, en particular, sobre redes neuronales convolucionales, estas ultimas siendo claves en el desarrollo de los algoritmos de reconocimiento facial actuales.

Una vez sentada una base teórica sólida, se presentaron definiciones precisas sobre las dos ramas principales dentro del reconocimiento facial: identificación y verificación. Se remarcaron sus diferencias y se analizaron las etapas claves que componen estos procesos. A partir de esta noción mas avanzada acerca del tema, se explico el uso de las redes neuronales convolucionales en la tarea especifica de reconocimiento facial y como podría ser beneficioso el uso del algoritmo de Triplet Loss.

En nuestros experimentos, se intentaron replicar los pasos de [19], sin embargo y tal como se puede apreciar en el Cuadro 4.3, no se pudieron alcanzar los mismos resultados. De nuestras pruebas y, basándonos en apreciaciones de trabajos similares ([9, 19, 24]), concluimos que la etapa de selección de triplets es clave en el entrenamiento de el algoritmo Triplet Loss. Tal como se explica en 3.4.2, la cantidad de posibles triplets a ser generados crece significativamente a medida que se incrementa el conjunto de datos de entrenamiento. Utilizar todos estos datos puede ser ineficiente, teniendo en cuenta además, que muchos de estos triplets pueden no contribuir al proceso de "aprendizaje". Por otro lado, es necesario respetar un balance en la búsqueda de muestras que sean informativas para nuestro modelo pero cuidando de no converger a mínimos locales en etapas tempranas del entrena-

miento. Dadas estas condiciones, la tarea de desarrollar un algoritmo de selección es critica, pudiendo conllevar además a la implementación de métodos poco eficientes.

MEJORAS Y TRABAJO FUTURO

A partir de este punto, es necesaria la investigación y prueba de nuevos algoritmos para el proceso de selección de triplets, evaluando su complejidad, pero principalmente analizando como se ve afectado y que variaciones se producen en las respuestas de el modelo de Triplet Loss, esperando así, obtener pistas que faciliten y logren mejores resultados durante su entrenamiento.

BIBLIOGRAFÍA

[1] P. N. Belhumeur, J. P. Hespanha y D. J. Kriegman. "Eigenfaces vs. Fisherfaces: recognition using class specific linear projection". En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.7 (jul. de 1997), págs. 711-720. DOI: 10.1109/34.598228 (vid. pág. 3).

[2] Enrico Blanzieri y Anton Bryl. "A survey of learning-based techniques of email spam filtering". En: *Artificial Intelligence Review* 29.1 (2008), págs. 63-92 (vid. pág. 5).

[3] Eugene Borovikov. "A survey of modern optical character recognition techniques". En: *arXiv preprint arXiv:1412.4183* (2014) (vid. pág. 5).

[4] Sumit Chopra, Raia Hadsell y Yann LeCun. "Learning a similarity metric discriminatively, with application to face verification". En: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, págs. 539-546 (vid. pág. 3).

[5] Ramazan Gokberk Cinbis, Jakob Verbeek y Cordelia Schmid. "Unsupervised metric learning for face identification in TV video". En: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, págs. 1559-1566 (vid. pág. 3).

[6] George Cybenko. "Approximation by superpositions of a sigmoidal function". En: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (1989), págs. 303-314 (vid. pág. 13).

[7] Navneet Dalal y Bill Triggs. "Histograms of oriented gradients for human detection". En: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, págs. 886-893 (vid. pág. 26).

[8] Xiaolong Fan y Brijesh Verma. "A comparative experimental analysis of separate and combined facial features for GA-ANN based technique". En: *Computational Intelligence and Multimedia Applications, 2005. Sixth International Conference on*. IEEE. 2005, págs. 279-284 (vid. pág. 3).

[9] Alexander Hermans, Lucas Beyer y Bastian Leibe. "In defense of the triplet loss for person re-identification". En: *arXiv preprint arXiv:1703.07737* (2017) (vid. pág. 37).

[10] Gary B Huang, Manu Ramesh, Tamara Berg y Erik Learned-Miller. *Labeled faces in the wild: A database for studying face recognition in unconstrained environments*. Inf. téc. Technical Report 07-49, University of Massachusetts, Amherst, 2007 (vid. pág. 31).

[11] Diederik P Kingma y Jimmy Ba. "Adam: A method for stochastic optimization". En: *arXiv preprint arXiv:1412.6980* (2014) (vid. pág. 33).

[12] P Latha, L Ganesan y S Annadurai. "Face recognition using neural networks". En: *Signal Processing: An International Journal (SPIJ)* 3.5 (2009), págs. 153-160 (vid. pág. 3).

[13] Erik Learned-Miller, Gary B Huang, Aruni RoyChowdhury, Haoxiang Li y Gang Hua. "Labeled faces in the wild: A survey". En: *Advances in face detection and facial image analysis*. Springer, 2016, págs. 189-248 (vid. pág. 4).

[14] Jingtuo Liu, Yafeng Deng, Tao Bai, Zhengping Wei y Chang Huang. "Targeting ultimate accuracy: Face recognition via deep embedding". En: *arXiv preprint arXiv:1506.07310* (2015) (vid. pág. 4).

[15] Chaochao Lu y Xiaoou Tang. "Surpassing Human-Level Face Verification Performance on LFW with GaussianFace." En: *AAAI*. 2015, págs. 3811-3819 (vid. págs. 1, 4).

[16] Warren S McCulloch y Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". En: *The bulletin of mathematical biophysics* 5.4 (1943), págs. 115-133 (vid. pág. 10).

[17] Seiichi Nakagawa. "A survey on automatic speech recognition". En: *IEICE TRANSACTIONS on Information and Systems* 85.3 (2002), págs. 465-486 (vid. pág. 5).

[18] Shaoning Pang, Daijin Kim y Sung Yang Bang. "Face membership authentication using SVM classification tree generated by membership-based LLE data partition". En: *IEEE transactions on Neural networks* 16.2 (2005), págs. 436-446 (vid. pág. 3).

[19] O. M. Parkhi, A. Vedaldi y A. Zisserman. "Deep Face Recognition". En: *British Machine Vision Conference*. 2015 (vid. págs. 31-34, 36, 37).

[20] Omkar M Parkhi, Karen Simonyan, Andrea Vedaldi y Andrew Zisserman. "A compact and discriminative face track descriptor". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2014, págs. 1693-1700 (vid. pág. 4).

[21] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman y col. "Deep Face Recognition." En: *BMVC*. Vol. 1. 3. 2015, pág. 6 (vid. pág. 28).

[22] Guido Rossum. *Python Reference Manual*. Inf. téc. Amsterdam, The Netherlands, The Netherlands, 1995 (vid. pág. 33).

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein y col. "Imagenet large scale visual recognition challenge". En: *International Journal of Computer Vision* 115.3 (2015), págs. 211-252 (vid. pág. 4).

[24] Florian Schroff, Dmitry Kalenichenko y James Philbin. "Facenet: A unified embedding for face recognition and clustering". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, págs. 815-823 (vid. págs. 4, 28, 36, 37).

[25] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot y col. "Mastering the game of Go with deep neural networks and tree search". En: *Nature* 529.7587 (2016), págs. 484-489 (vid. pág. 6).

[26] K. Simonyan y A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". En: *ArXiv e-prints* (sep. de 2014). arXiv: 1409.1556 [cs.CV] (vid. pág. 32).

[27] Yi Sun, Yuheng Chen, Xiaogang Wang y Xiaoou Tang. "Deep learning face representation by joint identification-verification". En: *Advances in neural information processing systems*. 2014, págs. 1988-1996 (vid. págs. 4, 27, 28).

[28] Yi Sun, Ding Liang, Xiaogang Wang y Xiaoou Tang. "Deepid3: Face recognition with very deep neural networks". En: *arXiv preprint arXiv:1502.00873* (2015) (vid. pág. 4).

[29] Yi Sun, Xiaogang Wang y Xiaoou Tang. "Deep Learning Face Representation from Predicting 10,000 Classes". En: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Jun. de 2014 (vid. págs. 4, 27, 28).

[30] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. Rabinovich. "Going Deeper with Convolutions". En: *ArXiv e-prints* (sep. de 2014). arXiv: 1409.4842 [cs.CV] (vid. pág. 28).

[31] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato y Lior Wolf. "Deepface: Closing the gap to human-level performance in face verification". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2014, págs. 1701-1708 (vid. págs. 4, 27, 36).

[32] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato y Lior Wolf. "Web-scale training for face identification". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, págs. 2746-2754 (vid. pág. 36).

[33] Matthew Turk y Alex Pentland. "Eigenfaces for Recognition". En: *Journal of Cognitive Neuroscience* 3.1 (1991). PMID: 23964806, págs. 71-86. DOI: 10.1162/jocn.1991.3.1.71. eprint: https://doi.org/10.1162/jocn.1991.3.1.71. URL: https://doi.org/10.1162/jocn.1991.3.1.71 (vid. pág. 3).

[34] Paul Viola y Michael J Jones. "Robust real-time face detection". En: *International journal of computer vision* 57.2 (2004), págs. 137-154 (vid. pág. 26).

[35] S. van der Walt, S. C. Colbert y G. Varoquaux. "The NumPy Array: A Structure for Efficient Numerical Computation". En: *Computing in Science Engineering* 13.2 (mar. de 2011), págs. 22-30. DOI: 10.1109/MCSE.2011.37 (vid. pág. 33).

[36] S. Yang, Y. Xiong, C. Change Loy y X. Tang. "Face Detection through Scale-Friendly Deep Convolutional Networks". En: *ArXiv e-prints* (jun. de 2017). arXiv: 1706.02863 [cs.CV] (vid. pág. 26).

[37] Erjin Zhou, Zhimin Cao y Qi Yin. "Naive-deep face recognition: Touching the limit of LFW benchmark or not?" En: *arXiv preprint arXiv:1501.04690* (2015) (vid. pág. 28).