TÉCNICAS DE RECONOCIMIENTO FACIAL Y MEJORAS CON LA IMPLEMENTACIÓN DE REDES NEURONALES

UNIVERSIDAD COMPLUTENSE DE MADRID

GRADO EN INGENIERÍA MATEMÁTICA

TRABAJO FIN DE GRADO

Miguel Sainz García

Cotutores: Antonio López Montes, Antonio Martínez Raya, María Teresa Benavent Merchán

FACULTAD DE MATEMÁTICAS

Departamento de Análisis Matemático y Matemática Aplicada

Curso: 2022/2023

28 de Junio de 2023

Resumen

El reconocimiento facial lleva siendo un ámbito de estudio desde la década de 1960, logrando grandes avances gracias al desarrollo de la computación en las décadas siguientes. Todo lo contrario a lo que pasó con la inteligencia artificial, que sufrió un estancamiento durante esos años. Sin embargo, gracias a los recientes avances en la inteligencia artificial y al desarrollo de nuevas tecnologías, el reconocimiento facial ha logrado pasar de un sistema tan solo al alcance de muy pocos, a algo que podemos encontrar en nuestro bolsillo.

En este trabajo se buscan exponer las técnicas más importantes de reconocimiento facial: tanto las primeras (basadas en subespacios vectoriales) como las más modernas, que incorporan la inteligencia artificial, con el fin de determinar la técnica más efectiva.

Palabras clave

Reconocimiento facial, Análisis de Componentes Principales (PCA), Anáslisis Discriminante Lineal (LDA), Inteligencia Artificial (IA), Redes Neuronales, Redes Neuronales Convolucionales (CNN).

Abstract

Facial recognition has been an area of study since the 1960s, achieving great advancements thanks to the development of computing in the following decades. In contrast, artificial intelligence experienced a stagnation during those years. However, thanks to recent advancements in artificial intelligence and the development of new technologies, facial recognition has gone from being a system only available to a select few to something we can find in our pockets.

This work aims to develop the most important facial recognition techniques, including both the early ones (based on vector subspaces) and the most modern ones, which incorporate artificial intelligence, in order to determine the best technique.

Keywords

Facial Recognition, Principal Component Analysis (PCA), Linear Discriminant Analysis (LDA), Artificial Inteligence (IA), Artificial Neural Networks, Convolutional Neural Networks (CNN).

Índice de figuras

Índice

1 Introducción al reconocimiento facial

En el ámbito del reconocimiento de caras humanas, el objetivo es identificar un rostro usando exclusivamente sus rasgos faciales. Otros aspectos relacionados son la identificación del sexo o la edad de una persona, o la generación de modelos de texturas para su uso en gráficos.

El reconocimiento facial es un sistema de identificación biométrico mediante el cual se reconoce la forma y estructura única del rostro. Todos los sistemas de reconocimiento facial capturan una imagen bidimensional o tridimensional de la cara de una persona, y luego comparan la información clave de esa imagen con una base de datos de imágenes conocidas.

1.1 2D vs 3D

Actualmente, la mayoría de los sistemas de reconocimiento facial dependen de imágenes en 2D, ya que la mayoría de dispositivos de captura de imagen no tienen la capacidad de capturar información de profundidad, y las propias bases de datos de referencia consisten de imágenes en 2D, como fotos policiales o de pasaporte.

En el reconocimiento facial 2D se utilizan principalmente puntos de referencia como la nariz, la boca y los ojos para identificar un rostro, y mide tanto el ancho y la forma de los rasgos, como la distancia que hay entre ellos en la cara, mientras que en el **3D se añade profundidad a la imagen**, permitiendo incluir el contorno de la cara o el tamaño de la nariz.

Además de las mejoras que proporciona el reconocimiento facial 3D por el simple hecho de tener en cuenta un mayor número de características, puede ayudar a mitigar uno de los problemas provocados por las condiciones de la captura de foto: la distorsión del rostro.

Este problema consiste en una deformación de angular: la luz converge sobre el sensor con un gran ángulo de incidencia, provocando una serie de distorsiones. Es provocado por diversos factores, como la distancia de captura, el ángulo de la toma o el tipo de lente usado. Se puede apreciar un ejemplo en la siguiente figura:

Figure 1: Distorsión del rostro [7]

1.2 Aplicaciones

Se pueden dividir las aplicaciones del reconocimiento facial en dos grandes categorías:

- **Identificación o reconocimiento de caras:** compara la imagen de una cara desconocida con todas las imágenes de caras conocidas que se encuentran en la base de datos para determinar su identidad. Generalmente el número de diferentes sujetos en la base de datos es muy elevado y se buscan coincidencias aunque sean mínimas.

  Algunas aplicaciones del reconocimiento facial son: monitorización en búsqueda de enfermedades, identificación en aeropuertos, videovigilancia o investigaciones policiales.

- **Verificación o autentificación de caras:** compara una imagen de la cara con otra imagen con la cara de la que queremos saber la identidad. El sistema confirmará o rechazará la identidad de la cara, por lo que habrá que exigir mayor confianza en el resultado obtenido, además el número de sujetos en la base de datos suele ser más bajo, por ejemplo una familia.

  El mayor uso de sistemas de verificación de caras se da en accesos restringidos basados en el rostro: ya sea para móviles, coches, hogares...

En el trabajo me centraré en el propio reconocimiento facial, no en verificación, aunque las diferencias en el funcionamiento son pequeñas.

1.3 Detección de objetos

Una disciplina en ocasiones confundida con el reconocimiento facial, es la detección de rostros. El objetivo de la detección de objetos es clasificar las diferentes regiones de una imagen de acuerdo a ciertas características distintivas. En el caso particular de la detección de rostros, se busca distinguir dentro de una

imagen si hay una cara o no, y dónde está.

El procedimiento general consiste en hacer encajar el rostro en un rectángulo de cierto tamaño prefijado (bounding box), y utilizar el mismo para todas las imágenes. De esta forma se consigue mitigar el efecto producido por la distancia a la que es tomada la fotografía, aunque haya otros efectos provocados por esta misma causa como la distorsión del rostro que se mencionó en el apartado 1.1. Otras técnicas más sofisticadas buscan obtener todo el contorno del rostro, utilizando técnicas de segmentación.

A la hora de resolver un problema de reconocimiento facial, es muy común tener que primero localizar el rostro en la imagen, ya que la distancia a la que se encuentra una persona y el fondo pueden incidir gravemente en la precisión de la técnica utilizada.

Figure 2: Ejemplo de detección de rostro

En la sección 4 será necesario aplicar técnicas de detección de objetos en uno de los problemas a resolver. En particular, se usará el detector incorporado en Matlab: vision.CascadeObjectDetector, que utiliza el algoritmo de Viola-Jones. Se puede leer más acerca del funcionamiento de este algoritmo en **I Algoritmo de Viola-Jones para la detección de objetos**.

1.4 Visión artificial

Todo lo mencionado anteriormente se puede englobar en la **visión por ordenador**: un grupo de tecnologías o herramientas que permiten a los equipos captar imágenes del mundo real, procesarlas y generar toda la información posible a través de ellas. Pero otra disciplina relacionada con el reconocimiento de imágenes es la visión artificial.

Esta se basa en emular la visión humana, es decir, conseguir que las máquinas puedan percibir y comprender una o varias imágenes, y actuar en consecuencia.

Por lo tanto, aunque los resultados que se buscan en esta disciplina y en la visión por ordenador son los mismos, la diferencia reside en que en la **visión artificial prima la eficiencia**, ya que se requiere de una reacción en pocos instantes.

Algunas aplicaciones son:

- **Industria:** es el campo donde es más usada, para realizar el control de calidad en procesos muy controlados, que aquí si son posibles de obtener, por lo que se logran muy buenos resultados.
- **Automoción:** desde hace años se persigue el coche autónomo, y la clave para lograrlo es lograr un sistema de visión artificial eficaz, aunque también disponen de otros sensores como ultrasonidos o láser.
- **Robótica:** lograr máquinas que no solo sean capaces de hacer un único trabajo programado, sino de entender el objeto que tiene delante, posibles fallos, etc y arreglarlo.
- **Medicina:** Además de analizar enfermedades sin necesidad humana, se trabaja para lograr convertir lo que se ve a través de una cámara en impulsos eléctricos que se enviarían al córtex visual, permitiendo así la visión a personas que no pueden ver.

2 Técnicas de Reconocimiento Facial

2.1 Estado del Arte

El problema del reconocimiento automático de rostros ha evolucionado a pasos agigantados desde que empezó a estudiarse en la década de los 70.

Durante los años 90 y principios de los 2000, las técnicas de reconocimiento facial basadas en teorías estadísticas lograron grandes resultados para imágenes frontales adquiridas en entornos controlados. Por lo tanto, actualmente el mayor reto consiste en hacer frente a la variabilidad en las caras cuando la imagen no está controlada.

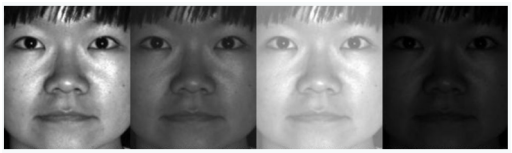
En esta sección, explicaré las técnicas de reconocimiento facial 2D más relevantes, comenzando por las más antiguas e intuitivas basadas en rasgos locales, y después otras técnicas más abstractas basadas en técnicas de la estadística y representar la imagen en subespacios vectoriales. También se hará una pequeña introducción al reconocimiento facial 3D.

2.1.1 Problemas y dificultades para el reconocimiento facial

Los problemas más comunes que se dan en el proceso de reconocimiento facial tienen que ver con la **base de datos**, y es que para un mismo rostro podemos tener una variación en: **la pose, algunos rasgos** (inclusión o no de gafas, tener o no bello facial, etc) y especialmente en **la iluminación**.

En algunas de las técnicas presentadas en la siguiente sección se mostrará cuáles de estos factores trae más complicación y su aproximación para tratar de evitar el problema, mientras que para otras no se tendrá tanto en consideración, pues están más pensadas para datasets controlados.

(a) Diferencias en la pose y rasgos corporales



(b) Misma cara, diferente iluminación [13]

Figure 3: Dificultades para reconocimiento facial

2.2 Técnicas de reconocimiento Facial

En general, todos los sistemas utilizan la misma secuencia de etapas para la identificación o verificación:

- Determinar un conjunto de **características independientes** para representar una cara.
- Representar las caras de **entrenamiento** en función de los valores que toman en ellas el conjunto de características seleccionado.
- Determinar estos valores para una **cara nueva** (desconocida).
- Emparejar la nueva cara con la más parecida de las de entrenamiento según cierto criterio: **clasificación**.

Tratamiento de la imagen:

En general, la imagen será tratada como una matriz de dimensiones $m \times n$, donde cada elemento se corresponde a un píxel. La imagen puede estar en color

o en escala de grises (al ser rostros no consideraremos imágenes en blanco y negro). Para las técnicas explicadas en esta sección se asumirá que están siempre en **escala de grises**, y sino serán convertidas. Por lo tanto, el valor de cada elemento de la matriz será un valor comprendido entre 0 (blanco) y 255 (negro).

Una imagen en color estará comprendida de 3 matrices con los mismos posibles valores, pero en este caso harán referencia a la cantidad de color Rojo, Verde y Azul, respectivamente. La conversión de una imagen en color a escala de grises se suele hacer mediante la **fórmula NTSC**. Se denomina así por National Television System Committee, el sistema de televisión analógico empleado en gran parte del mundo. Se busca representar la percepción relativa de la persona promedio del brillo de la luz roja, verde y azul. El valor en escala de grises es obtenido redondeando a la unidad la siguiente operación.

$$0.299 \times \text{Rojo} + 0.587 \times Verde + 0.114 \times Azul$$

2.2.1 Características geométricas

Las **primeras técnicas** de reconocimiento facial que trataron de implementarse fueron basadas en el cálculo de características geométricas.

Estas características son ángulos, relaciones o distancias en la cara. Son obtenidas sin discriminar por características individuales del rostro como ojos, boca o nariz, por ello la información que queda es puramente geométrica, y se representa por un vector numérico para los principales rasgos.

El procedimiento es el siguiente: se escoge un número de puntos asociados a lugares característicos de una cara, como pueden ser la nariz, el labio, los ojos, etc. De las m imágenes a clasificar, dada la i-ésima imagen de entrenamiento, asumimos que tenemos n_i muestras de ella. Para cada una de ellas, se buscan todos estos puntos en cada cara y se calculan ciertas distancias, relaciones entre distancias, áreas o ángulos entre ellos, que se almacenan en un **vector** de dimensión k, de tal forma que tenemos una serie de vectores $\vec{v_i}^j \in \mathbb{R}^k$ con $j \in \{1, ..., n_i\}$. Es importante que los valores estén normalizados de alguna forma, por ejemplo divididos entre su máximo, pues sino, el valor de alguna medida puede tener más peso que el de otra.

Como **representante** de la clase i, se toma la media de estos vectores: $\vec{v_i} = \frac{1}{n_i} \sum_{j=1}^{n_i} \vec{v_i}^j$. Dada una nueva imagen de test, se calculan de igual forma las mismas medidas, que se almacenan en un vector $\vec{w}$. Esta nueva imagen se clasificará como la imagen de entrenamiento tal que se minimice la **distancia euclídea** entre el vector que representa a la clase y a la nueva imagen:

$$argmin_{i \in \{1,...,m\}} ||v_i - w||^2 = argmin_{i \in \{1,...,m\}} \sum_{j=1}^{k} (v_{ij} - w_j)^2$$

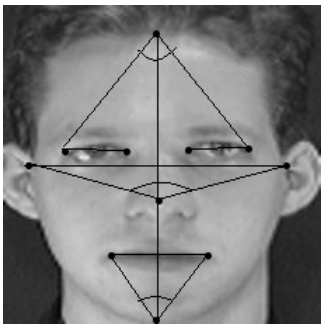
En un principio, los métodos utilizaban muchos puntos y pocas características asociadas a ellos, pero los mejores resultados se obtuvieron al tomar **pocos puntos** pero todas las características posibles entre ellos.

Ejemplo de aplicación

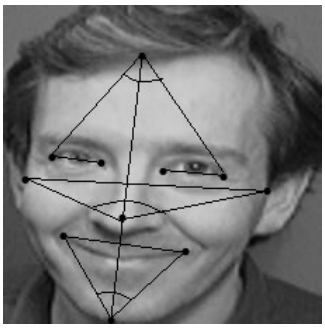
Voy a tomar como puntos de referencia el mentón, la punta de la nariz, el punto de la frente correspondiente a la recta que une ambos, los extremos tanto de los ojos como de la boca, y los puntos superiores de unión con las orejas. De estos puntos, voy a tomar las siguientes relaciones, que se apreciarán mejor al ver la imagen:

- Longitud horizontal de la boca y media de la longitud de ambos ojos entre el ancho de la cara
- Ancho de la cara entre largo de la cara
- Ángulo formado por: la boca respecto al mentón, los ojos respecto al centro de la frente, y la nariz frente a las orejas, divididos entre $180^{\underline{o}}$ para estandarizarlos.

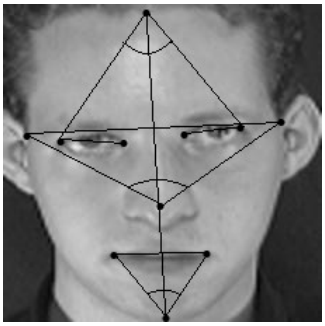
Se han medido estas relaciones en las siguientes imágenes:



(a) sujeto 1



(b) sujeto 2



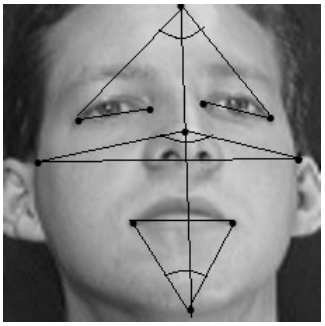


Figure 5: Ejemplo de características geométricas

Los vectores obtenidos son los siguientes:

$$v_1 = \begin{bmatrix} 0.37 \\ 0.25 \\ 0.9 \\ 0.42 \\ 0.83 \\ 0.44 \end{bmatrix} \quad v_2 = \begin{bmatrix} 0.5 \\ 0.5 \\ 0.9 \\ 0.42 \\ 0.82 \\ 0.44 \end{bmatrix} \quad v_3 = \begin{bmatrix} 0.37 \\ 0.25 \\ 0.84 \\ 0.39 \\ 0.64 \\ 0.42 \end{bmatrix} \quad v_4 = \begin{bmatrix} 0.4 \\ 0.27 \\ 0.86 \\ 0.33 \\ -0.65 \\ 0.47 \end{bmatrix}$$

Así, si queremos clasificar las dos últimas imágenes en función de las dos primeras, tenemos que:

$$||v_3 - v_1||^2 = 0.041 \quad ||v_3 - v_2||^2 = 0.117$$

$$||v_4 - v_1||^2 = 2.201 \quad ||v_4 - v_2||^2 = 2.234$$

Se ve que ambas serían clasificadas correctamente como el primer sujeto, sin embargo la última imagen tiene un valor muy distinto a todas las demás en el ángulo de la nariz con las orejas, simplemente por el ángulo desde el que ha sido tomada la imagen. Esto podría llevar a mayores problemas, y es que esta técnica puede producir resultados muy malos cuando se aplica a una base de datos no controlada.

En sus inicios, la búsqueda de estos puntos en cada rostro requería un proceso **manual**. A día de hoy hay técnicas que permiten automatizar esta parte del proceso, pero esta técnica ha quedado en desuso por ofrecer peores resultados que otras que llegaron más adelante, algunas basadas en la misma idea.

2.2.2 Correspondencia de plantillas

Esta técnica mejoró los resultados de las características geométricas con un enfoque similar pero con mayor número de puntos y por lo tanto mayor coste computacional. Consiste en realizar una máscara o plantilla de la cara para después compararla con la nueva imagen. Utilizando ciertos puntos generales, se fuerza que las plantillas coincidan en ellos y se calcula la diferencia entre el resto de puntos de ambas plantillas.

Hasta aquí no hay mucha diferencia con la técnica anterior, la mejora vino cuando se incorporaron las **plantillas por partes**: de boca, ojos, nariz, etc. Por ejemplo la plantilla para el ojo tiene 11 parámetros: arco superior e inferior del ojo, círculo del iris, puntos del centro del iris y de la parte blanca de los ojos y ángulos de inclinación del ojo, entre otros.

Esto permite incorporar mayor información y encontrar de forma más detallada dónde se producen las diferencias entre dos rostros. Por ejemplo, si dos rostros coinciden en todo salvo en la nariz, puede que haya sufrido un golpe allí, o el ángulo de la foto haya cambiado lateralmente. Además, las plantillas son **deformables**. Esto quiere decir que se pueden trasladarse, rotar y cambiar de tamaño para ajustarse a la representación de su forma presente en la imagen.

Con esta mejora puede dar buenos resultados pero es **computacionalmente aún más costoso**. Además, al igual que con el anterior método, esta técnica es eficaz cuando las condiciones de test son muy similares o iguales a las de entrenamiento, y sigue requiriendo un gran proceso previo manual, por lo que tampoco me centraré en ella.

Figure 6: Correspondencia de plantillas [14]

2.2.3 Correspondencia entre agrupaciones de grafos elásticos: EBGM

EBGM (Elastic Bunch Graph Matching) tiene en cuenta que las imágenes faciales reales tienen muchas características no lineales que no son tratadas en los métodos lineales de análisis discutidos previamente, tales como variaciones en la iluminación (Iluminación de exteriores vs. Interior fluorescente), postura (frontal vs. inclinada) y expresión (sonrisa vs. ceño fruncido).

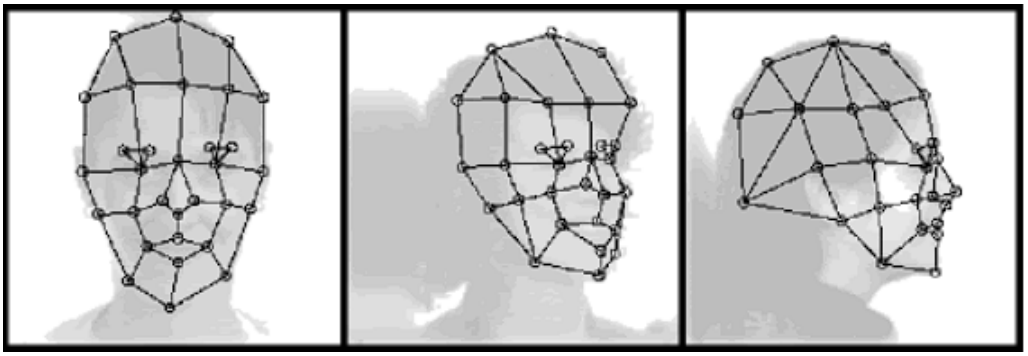


Figure 7: Variación de postura con EGBM [14]

Su funcionamiento sigue el de los métodos más tradicionales, basados en la selección de ciertos puntos de un rostro que se han mencionado anteriormente, siendo el más complejo y que mejor resultado ha dado de los que tienen este enfoque. Estos puntos constituirán los nodos del **grafo del rostro** para una imagen, pero para conseguir que se pueda adaptar a todas las variaciones mencionadas previamente, se crea una **agrupación de grafos**. Esto es, se aplica el grafo a una nueva imagen y se corrige manualmente cada nodo que no haya quedado correctamente posicionado.

Con ambos grafos se realiza esto para sucesivas imágenes hasta obtener un resultado convincente. Para el resto de imágenes de entrenamiento se aplican los grafos de forma automática, para generar la **galería de grafos**, y así con cada rostro que queramos en la base de datos hasta tenerlos todos. El método que se utiliza para poder reconocer rostros bajo condiciones variables es crear distintas galerías para una sola imagen, separándolas según pose, iluminación, etc, de tal forma que se producirá semejanza con tan solo una de ellas.

Dada una nueva imagen, se crea su grafo de la misma forma, y se compara con cada galería de grafos para ver a cuál se asemeja más. El cálculo de la semejanza entre dos grafos se hace mediante una **función de similaridad:** esta podría ser usar las coordenadas de los nodos en los ejes vertical y horizontal para medir la distancia relativa de los obtenidos a los de cada grafo, sin embargo, esto es computacionalmente costoso, así que se presenta la siguiente heurística, que es la que le da el nombre de elástico al método. Se basa en 3 movimientos:

- **Movimiento Global:** Escanear la imagen para encontrar la posición del objeto. Se desplaza el grafo por toda la cuadrícula de píxeles de la imagen aprovechando que las distancias relativas entre nodos se mantienen constantes, así que solo importan las similitudes Jet, que están explicadas más adelante.
- **Movimiento de escala:** Se amplía o disminuye el tamaño del grafo horizontal y verticalmente para mejorar la similitud con el grafo modelo. Esto está basado en que la imagen puede estar tomada desde más o menos cerca, y como la posición global se ha buscado previamente, no producirá a priori errores de sobreajuste a algunos nodos.
- **Movimiento local:** Se desplazan ligeramente los nodos de forma aleatoria para adaptarse a las distorsiones o diferencias locales, de ahí la elasticidad.

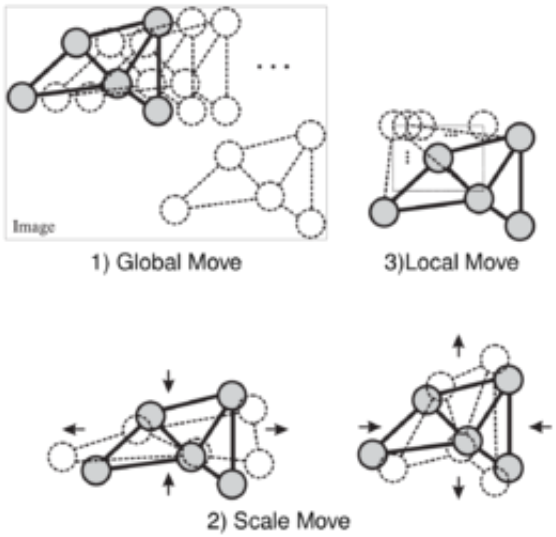


Figure 8: EGBM: Esquema heurístico para correspondencia de grafos elásticos [9]

Si tras este proceso no se ha encontrado una similitud conveniente, se descarta que la imagen provenga de la misma persona que el modelo de entrenamiento y se pasa al siguiente. Lo normal es que este algoritmo proporcione varias coincidencias: si los grafos obtenidos como más similares son de la misma galería, tendremos certeza de que se trata de esa persona, pero si se obtienen distintas galerías, se podrán identificar características acerca de esa persona sin necesidad de verla, como su género, edad, etc, lo cual puede ser útil para otros problemas. También se puede dar el caso de que no haya coincidencias, que puede significar que el rostro no se encontraba entre los sujetos de prueba, o que hay variaciones demasiado significativas.

El algoritmo se basa en las **ondículas de transformación Gabor:** crea una arquitectura de enlace dinámico que proyecta el rostro sobre la planilla elástica. El **Jet Gabor** es un nodo en la planilla elástica, manifestado por círculos en la imagen debajo, el cual describe el comportamiento de la imagen alrededor de un píxel. Este funcionamiento utiliza una función de convolución, que como explicaré más adelante en la sección acerca de redes neuronales convolucionales, busca extraer las características de la imagen mediante operaciones a un conjunto de píxeles.

Las ondículas de Gabor permiten obtener la resolución conjunta óptima de información en los espacios bidimensionales espacial y frecuencial: representan y almacenan la frecuencia de la información de una región específica de la imagen. Por ello, se utilizarán como filtros que se aplicarán a los nodos del grafo de una imagen. Esto es mediante las siguientes ecuaciones de onda que

las separan en parte par e impar:

$$\text{Par } Ond_p(x,y) = e^{\frac{-(x-x_0)^2+\gamma^2(y-y_0)^2}{2\sigma^2}} cos(\frac{2\pi x_r}{\lambda})$$
$$\text{Impar } Ond_i(x,y) = e^{\frac{-(x-x_0)^2+\gamma^2(y-y_0)^2}{2\sigma^2}} sen(\frac{2\pi x_r}{\lambda})$$

donde λ define la longitud de onda, σ define el radio de la gaussiana, γ define la relación de aspecto de la gaussiana y (x_0, y_0) determinan la ubicación del valor pico de la función gaussiana.

La función de similitudes Jet es similar a una correlación, y se define como:

$$S(J,J') = \frac{\sum_{j=1}^{n} a_j a'_j cos(\phi_j - \phi'_j)}{\sqrt{\sum_{j=1}^{n} a_j^2 \sum_{j=1}^{n} a_j'^2}}$$

donde a' es la magnitud del Jet de la base de datos, a la magnitud del Jet de la imagen nueva, ϕ'_j la fase del Jet de la base de datos, ϕ_j la fase del Jet de la imagen nueva y n el número de coeficientes del Jet.

Con estas definiciones, el movimiento global quedaría definido de la siguiente forma:

- **1)** Se aplican a modo de filtro distintas ondículas de Gabor a los nodos de la imagen, que darán como respuesta unos valores que se denominan Jet, caracterizados por una fase ϕ_j y una magnitud a.
- **2)** Para la nueva imagen se calcula la función de similitud Jet tal y como se acaba de ver. Con esto se deduce qué imágenes son más similares en términos relativos a la nueva imagen, y de ese conjunto reducido, con los siguientes movimientos, se clasifica como una de ellas.

2.2.4 PCA

El **análisis de componentes principales** está basado en la descomposición SVD, el objetivo es calcular la matriz de correlaciones normalizada y diagonalizarla. Por ello, primero explicaré en qué consiste la descomposición SVD.

La **descomposición en valores singulares (SVD)** es una factorización de una matriz $A \in \mathbb{R}^{m \times n}$ real o compleja como producto de tres matrices $U\Sigma V^T$, donde en U se almacenan los autovectores de la izquierda, en V de la derecha, y en Σ los autovalores. Tienen las siguientes propiedades:

- U es una matriz cuadrada y unitaria , con dimensión $n \times n$, por lo que $U^T U = I_n$ y sus n columnas $u_1, ..., u_n$ forman una base de $\mathbb{R}^n$. Como cada columna de A: $a_1, ..., a_n$ está en el espacio $\mathbb{R}^n$, se tiene que las columnas de U forman una base para las columnas de A.

- Σ es una matriz de dimensión $n \times m$. Es pseudodiagonal, esto es, está formada por $p = min\{m, n\}$ elementos σ_i dispuestos de forma diagonal, y el resto ceros. Además, estos valores están ordenados, de tal forma que $\sigma_1 \geq ... \geq \sigma_p \geq 0$. Por lo tanto, cada uno de estos valores representa la importancia de las columnas de U y V asociadas a él a la hora de representar A, y permiten reducir el tamaño de estas matrices, utilizando solo los valores más elevados, para obtener una aproximación de A en un espacio menor. Por ejemplo, para $n > m$:

$$\Sigma = \begin{bmatrix} \sigma_1 & ... & 0 \\ \vdots & \ddots & \vdots \\ 0 & ... & \sigma_m \\ \\ 0 & ... & 0 \end{bmatrix}$$

- V, al igual que U, también es una matriz cuadrada y unitaria de dimensión $m \times m$. En su transpuesta V^T, la primera fila v_1 de esta matriz representa qué combinación de las n columnas de U hay que tomar para obtener la primera columna de A.

Ahora vamos a ver el proceso para obtener esta descomposición para el caso que nos interesa, que es cuando $n > m$, es decir, hay más columnas que filas, ya que la columnas representan el tamaño de la imagen (el número de píxeles), y las filas el número de imágenes. En ciertas bases de datos puede haber mayor número de sujetos que de píxeles, pero lo habitual es lo contrario.

Se parte de una matriz $A \in \mathbb{R}^{m \times n}$ tal que $n > m$:

$$A = \begin{bmatrix} a_{11} & ... & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & ... & a_{mn} \end{bmatrix}$$

Se calcula $A^T A$ de dimensión $n \times n$ cuadrada y simétrica, de la cual se calculan sus autovalores como los valores de λ tal que $|A - \lambda I_n| = 0$ y se ordenan de tal forma que $\lambda_1 \geq ... \geq \lambda_n \geq 0$, y sus respectivos autovectores unitarios como los v_i tal que $(A - \lambda_i I_n)\vec{v_i} = \vec{0}$ y $\vec{v_i}^T \vec{v_i} = 1$, que son ortogonales entre si. Al menos los últimos $n - m$ autovalores tendrán que ser nulos pues el rango de la matriz es como máximo m, así que la matriz Σ, cuya diagonal son las raíces de estos autovalores, queda de la siguiente forma:

$$\Sigma = \begin{bmatrix} \sqrt{\lambda_1} & ... & 0 \\ \vdots & \ddots & \vdots \\ 0 & ... & \sqrt{\lambda_m} \\ \\ 0 & ... & 0 \end{bmatrix}$$

La matriz V se construye a partir de los autovectores obtenidos, colocados por columnas $V = [\vec{v_1}, ..., \vec{v_n}]$. Y por lo tanto, la matriz U debe ser tal que: $u_i = \frac{Av_i}{\sigma_i}$ donde $\sigma_i > 0$, o en caso de no haber m autovalores positivos, tomando valores ortonormales entre si a los u_i anteriores.

Algunos de los autovalores pueden ser nulos, cuando esto pasa, se puede reducir la dimensión de las matrices de la descomposición ya que no se pierde información acerca de los autovalores, con el objetivo de reducir el coste computacional, en lo que se conoce como SVD economizado. También se pueden eliminar autovalores cercanos a cero, o como se verá a continuación, quedarse con las matrices truncadas en el rango r, donde r puede ser prefijado de antemano.

El objetivo entonces es obtener la mejor aproximación de A de rango r. El **Teorema de Eckard-Young** dice que de entre todas las matrices de rango r, la matriz $\tilde{A}$ que mejor aproxima A, viene dada por las primeras r columnas de las matrices U y V, y la matriz cuadrada de dimensión r correspondiente de Σ, que respectivamente se denotan como $\tilde{U}$, $\tilde{V}$ y $\tilde{\Sigma}$. Es decir, $argmin\{||A - \tilde{A}||\ \ tq\ \ rg(\tilde{A}) = r\} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$

Ahora sí, se expone la técnica de PCA:

Se parte de n observaciones $\vec{x_1} \dots \vec{x_n}$ con las que se calcula la siguiente matriz X:

$$X = \begin{bmatrix} \vec{x_1} \\ \vdots \\ \vec{x_n} \end{bmatrix}$$

Se trabaja con una matriz de **media nula**, para ello se calcula la media de la matriz anterior por columnas y se le resta para obtener:

$$\vec{\bar{x}} = \frac{1}{n} \sum_{j=1}^{n} \vec{x_j}$$

y con ello

$$\bar{X} = \begin{bmatrix} \vec{\bar{x}} \\ \vdots \\ \vec{\bar{x}} \end{bmatrix}$$

para así lograr la matriz de media nula: $B = X - \bar{X}$ para la cual se calcula la matriz de covarianza por filas: $C = B^T B$. Esta matriz es simétrica y se calculan sus **autovectores** y **autovalores**, que son almacenados por orden decreciente de la magnitud del valor absoluto del autovector en las matrices V y D, respectivamente, tal que V tiene los autovectores dispuestos por columnas y D es diagonal.

La matriz de componentes principales será $T = BV$, que además, dada la descomposición SVD de $B = U\Sigma V^T$, coincide con $T = U\Sigma$. Es muy común emplear la técnica de SVD para calcular estos valores debido a su eficiencia, y sobre todo a que siempre existe, en contra de la diagonalización de matrices, que puede existir o no.

Respecto a esta igualdad entre ambas descomposiciones, cabe añadir que cada autovalor de D coincide con un autovalor σ^2 de Σ. Además, podemos calcular el porcentaje de la variabilidad total ($\sum_{k=1}^{n} \lambda_k$) que se está almacenando en los primeros r autovectores como:

$$\frac{\sum_{k=1}^{r} \lambda_k}{\sum_{k=1}^{n} \lambda_k}$$

Lo que se busca es reducir la dimensionalidad, para también reducir el coste computacional de trabajar con matrices mayores. Para ello, de los n autovalores, el objetivo es quedarse tan solo con los r que cumplan cierto requisito, que puede ser:

- **1.** Todos aquellos cuyo autovalor asociado sea mayor que cierta precisión: Si $\lambda_i > prec$, se utilizará la columna correspondiente de V: v_i. Sino no.
- **2.** Todos aquellos tal el porcentaje acumulado de variabilidad total sea menor que lo buscado. Como están almacenados en orden decreciente, se van sumando λ_i hasta que $\frac{\sum_{k=1}^{r} \lambda_k}{\sum_{k=1}^{n} \lambda_k}$ supere cierto porcentaje, y entonces se para.
- **3.** Este último método es gráfico. Consiste en buscar dónde el diagrama de bastón (gráfica de la variabilidad explicada acumulada según se añaden autovalores) tiene un salto de tamaño más significativo, y quedarse con todos hasta ello.

Usando cualquiera de estos métodos, se obtiene una matriz $\tilde{V}$ reducida formada por solo los autovectores seleccionados. Se busca que sea unitaria, es decir formada por vectores columna ortonormales entre si (lo cual ya es por cómo ha sido calculada) y unitarios, para lo cual se divide cada columna entre su norma cuadrática.

Si se quieren recomponer las imágenes a partir de estos autovectores, se calcularían como $\tilde{X} = B\tilde{V}$, lo cual se denomina **Eigenfaces**, y después para cada imagen recomponer los vectores a sus dimensiones originales. A continuación se proyectan las imágenes originales en este nuevo espacio como $PX = \tilde{X}^T B = \tilde{V}^T C$.

Un ejemplo muy visual del funcionamiento es el siguiente: si se aplica la técnica de análisis de componentes principales a una **nube de puntos** con correlación lineal, se obtiene como característica más importante la dirección que sigue la nube, y como segunda característica la normal a ella.

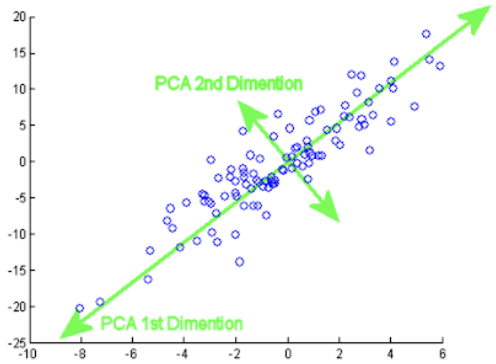


Figure 9: PCA de una nube de puntos [15]

Hasta ahora se ha expuesto el funcionamiento de la técnica general, lo siguiente es la aplicación particular al reconocimiento de imágenes: Se parte de un conjunto de imágenes de n rostros distintos, con m_i imágenes de cada uno de ellos. Estas imágenes vienen dadas como una matriz de píxeles, así que se convierten a vector ya sea por filas o columnas, pero se deberá mantener la elección para todas las imágenes. Así, la j-ésima imagen del rostro i viene dada como un vector $\vec{v_i}^j$.

A continuación, se agrupan todos los vectores asociados a una misma clase para tomar un único vector por rostro $\vec{v_i} = \frac{1}{m_i} \sum_{j=1}^{m_i} \vec{v_i}^j$, y se toma como matriz X la formada por el vector asociado a cada rostro: $X^T = [\vec{v_1}, ..., \vec{v_n}]$. Se aplica todo el proceso descrito anteriormente para obtener PX. Finalmente, dada una nueva imagen, se transforma vectorialmente como antes a un vector $\vec{w}$, al cual se le resta la misma media obtenida anteriormente para obtener $\tilde{w}$ y se

proyecta en el espacio como $Pw = \tilde{X}^T \tilde{w}$. Se clasificará como la clase con la que tenga menor distancia euclídea en este espacio: $argmin_{i=1,..,n}||Pw - PX_i||$.

2.2.5 LDA

El objetivo del Análisis Discriminante Lineal (LDA), es separar en clases diferentes de tal forma que queden completamente diferenciadas. Esto es, encontrar la proyección de tal modo que los determinantes de la matriz **entre-clase** (between class) se minimicen mientras que los de las matrices de dispersión **dentro de la clase** (within class) de las muestras proyectadas alcancen su máximo. Para lograr esto, se maximiza el **ratio** (la división) del segundo entre el primero. Esto permite reducir la importancia de factores externos, como la **iluminación**, al tratar de reconocer una imagen.

Su mayor dificultad es que el vector imagen generado es de una dimensionalidad muy elevada, lo cual, obviando el **coste computacional**, debería dar muy buenos resultados por la grandísima cantidad de información que utiliza, al contrario que el PCA, que reduce la dimensionalidad. Esto permite que el método encuentre la proyección de patrones óptima para la clasificación. Además, la idea detrás de calcular ese ratio es eliminar las variaciones que se producen entre imágenes debido a la iluminación o el gesto, que se corresponden a los autovalores más grandes, por ello suele ofrecer mejores resultados en unos datos de test no controlados o no similares a los de entrenamiento respecto a PCA.

El método consiste en lo siguiente: sean $\{X_i\}_{i=1}^N$ las imágenes de entrenamiento, C el número de clases a clasificar, C_i la i-ésima clase con n_i ejemplares. Las imágenes son de dimensión $m \times n$, y se proyectan en un **espacio vectorial** de menor dimensión, dado por cierto vector $\vec{\alpha}$: $\vec{y_i} = X_i \vec{\alpha}$. Además se denota por $\bar{y}_i$ a la media de la iésima clase, y por $\bar{y}$ a la media global.

La **medida escalar de Fisher para la separabilidad** de clases para cada posible valor de α es $J(\alpha) = \frac{tr(S_b^\alpha)}{tr(S_W^\alpha)}$ donde:

$$S_b^\alpha = \frac{1}{N} \sum_{i=1}^{C} ni(\bar{y}_i - \bar{y})(\bar{y}_i - \bar{y})^T$$

$$S_w^\alpha = \frac{1}{N} \sum_{i=1}^{C} \sum_{j \in C_i} (y_j - \bar{y}_i)(y_j - \bar{y}_i)^T$$

Para calcular esta medida, también se puede usar que $tr(S_b^\alpha) = \vec{\alpha^T} G_b \vec{\alpha}$ y

$tr(S_w^\alpha) = \vec{\alpha^T} G_w \vec{\alpha}$, donde

$$G_b^\alpha = \frac{1}{N} \sum_{i=1}^{C} ni(\bar{X}_i - \bar{X})(\bar{X}_i - \bar{X})^T$$

$$G_w^\alpha = \frac{1}{N} \sum_{i=1}^{C} \sum_{j \in C_i} (X_j - \bar{X}_i)(X_j - \bar{X}_i)^T$$

que se llaman **matrices de dispersión de entre clases y dentro de la clase**, respectivamente. De esta forma $J(\vec{\alpha}) = \frac{\vec{\alpha^T} G_b \vec{\alpha}}{\vec{\alpha^T} G_w \vec{\alpha}}$

El objetivo reside en encontrar α que maximice el ratio, este viene dado por $G_b \vec{\alpha} = \lambda G_w \vec{\alpha}$, una estructura particular de autovectores que proporcionará los k mayores autovectores: $\vec{\alpha_1} ... \vec{\alpha_k}$, que serán las direcciones ortogonales donde se proyectará la estructura, y que se almacenan en una matriz V por columnas.

Una vez hallados, se proyectan las imágenes de entrenamiento en media en este subespacio siguiendo el procedimiento visto para PCA (2.2.4): Se expresa vectorialmente la j-ésimo imagen del rostro i $\vec{x_i}^j$ y se calcula la media por clases $\vec{x_i}$ para dar lugar a la matriz X. A esta se le resta la media global en cada elemento para obtener la matriz B. Con esta se calcula $Y = BV$ y la proyección de las imágenes $U = Y^T B$. Dada una nueva imagen, se calcula análogamente su expresión vectorial, a la cual se le resta la misma media global calculada anteriormente para obtener $\vec{w}$, cuya proyección en el subespacio es $W = Y^T \vec{w}$. Se clasificará como la clase con la que tenga menor distancia euclídea en este espacio: $argmin_{i=1,..,n} ||W - U_i||$.

Sin embargo, el cálculo de los valores α_i tiene un coste computacional muy elevado, por lo que se buscan otras posibilidades de implementación reduciendo la dimensionalidad:

FLD o Fisherfaces

FLD (Fisher Discriminant Analysis), también conocido como Fisherfaces es la solución tradicional al problema. Primero se proyecta la matriz (imagen) sobre un espacio de menor dimensión utilizando PCA, para después poder aplicar LDA. Pero, hay un problema, la información que PCA considera relevante no tiene por qué coincidir con la que LDA considera, por lo que podría estar **perdiendo información clave**. El algoritmo consiste en lo siguiente:

- **1) PCA:** Aplicar la técnica de PCA tal y como se muestra en 2.2.4 para obtener la matriz B y las Eigenfaces, las cuales denotaré por Y de aquí en adelante.
- **2) Matrices de dispersión:** Calcular para las imágenes de entrenamiento proyectadas en el subespacio generado por Eigenfaces las ma-

trices de dispersión entre clases Sb y dentro de la clase Sw tal como se indica en esta misma sección.

- **3) Autovalores y Autovectores:** Calcular los autovalores de Sw, que se guardan en el vector NSw. También el autovalor de Sb, que solo tiene uno correspondiente a su único autovalor, que se almacena en un vector VSb.

- **4) Fisherface:** Se denominan así al vector obtenido mediante la operación $F = Y \cdot VSb \cdot NSw^T$. Para obtener los valores en este nuevo subespacio para cada rostro, se multiplica por B^T, obteniendo $U = F^T \cdot B$.

- **5) Imagen de Test:** Utilizando la proyección de la imagen en el espacio obtenido por Fisherface, $W = FY^T\vec{w}$, se clasificará como la clase con la que tenga menor distancia euclídea en este espacio: $argmin_{i=1,..,n}||W - U_i||$.

Se puede ver más acerca de esta técnica y este algoritmo en [9, 1].

2.3 Reconocimiento Facial 3D

En general, la proyección de un objeto tridimensional a dos dimensiones conlleva gran pérdida de información que sería muy útil para mejorar la precisión de estos sistemas. Esto motiva la aparición de técnicas de reconocimiento facial en 3 dimensiones, para las cuales primero se necesita **adquirir la imagen tridimensional**, para lo cual también se tienen distintas técnicas que se pueden clasificar en 3 grandes grupos:

- **Técnicas pasivas:** Denominadas así por el uso de sensores pasivos: cámaras que capturan imágenes en 2d. A partir de ellas se reconstruye la imagen utilizando la **Visión Estereoscópica**, o técnicas derivadas. Esta técnica recibe su nombre de la propia capacidad de la visión del ser humano de que a partir de dos imágenes ligeramente diferentes recibidas por cada ojo, el cerebro recomponga una única imagen total tridimensional. En ocasiones, para lograr esto se necesitan procesos manuales para indicar puntos comunes a ambas imágenes, previo a obtener el modelo 3D definitivo.

- **Técnicas activas:** Además de sensores pasivos, incorporan un sensor activo en alguna operación. Por ejemplo, en el ámbito de la visión artificial, se utilizan sensores de rango basados en la **Triangulación Activa**, en los cuales se incorpora un láser que se proyecta sobre el objeto. De esta forma se obtienen mapas densos y precisos de coordenadas 3D eliminando el apartado manual del anterior método

- **Imágenes de rango:** Utilizan digitalizadores 3D para directamente codificar la forma de la superficie del rostro, dando un mallado de puntos (x,y,z) con las relaciones entre ellos (generalmente una triangulación). Estos son muy útiles cuando la técnica de reconocimiento facial que se vaya a utilizar posteriormente vaya a utilizar un grafo o mallado de la superficie, sino, se está perdiendo información de la imagen, aunque algunos son capaces de obtener miles de puntos.

Las técnicas que se utilizan para reconocer rostros en 3D son las mismas que en 2D, aunque cambia la manera de recoger las características, ya que ahora no se dispone de una imagen con ejes (x, y) sino de un **espacio tridimensional** (x, y, z). Las técnicas basadas en las ya vistas se pueden dividir en 2 grupos:

- **Técnicas basadas en rasgos locales:** Especialmente utilizadas tras la aparición de los digitalizadores, que ya proporcionan el mallado de puntos. Ahora permiten incorporar características como la curvatura de la superficie o la textura, que ahora no se ven distorsionadas por cómo se haya tomado la imagen.

- **Técnicas basadas en el análisis de componentes principales:** Para mantener el enfoque matricial, no se utilizan las coordenadas (x,y,z), sino que se utilizan ciertas imágenes 2D de las obtenidas para calcular un conjunto de autovectores, y por separado, otro conjunto obtenido mediante los mapas de profundidad: una matriz que, a cada píxel le hace corresponder la distancia obtenida con el sensor. Finalmente, para una imagen de test, se evalúan ambas distancias y se busca la categoría que minimice ambas.

Se puede ver más acerca de estas técnicas en [4].

2.4 Clasificadores

Para la parte final del proceso de reconocimiento facial, la clasificación, se pueden utilizar múltiples clasificadores propios de la ciencia de datos como son:

- **Minimizar una distancia:** una vez se ha expresado la imagen en función de ciertas características, el método más antiguo es clasificarla como la imagen del conjunto de entrenamiento con menor distancia (por ejemplo euclídea) entre ellas.

- **Redes Neuronales:** se explicarán más adelante.

- **Máquinas de vector soporte (SVM):** muy útiles para problemas de clasificación binaria, especialmente si se trata de conjuntos linealmente

separables. Consiste en hallar el hiperplano que separa ambos conjuntos, o en caso de no existir ninguno (lo más común), el que minimice el error empírico.

- y otros métodos como **KNN, Análisis Discriminante o Regresión logística** muy comunes en la ciencia de datos.

2.4.1 El perceptrón multicapa como clasificador

El **perceptrón** es la unidad básica de una red neuronal, también es conocido como neurona pues su funcionamiento trata de imitar las del cerebro humano. Esto lo hace recibiendo como entradas n valores (un vector $\vec{x} \in \mathbb{R}^n$) los cuales opera con unos pesos w_i con $i = 0, .., n$ para dar lugar a una salida, que se usará para clasificar los datos de entrada de acuerdo a ciertas categorías, en este caso los rostros.

El **entrenamiento** consiste en recalcular los valores de estos pesos hasta dar con los valores que clasifiquen correctamente los datos del problema, por lo que se trata de aprendizaje supervisado. El algoritmo que se suele emplear para ello está basado en el método del gradiente descendiente. Sin embargo, el perceptrón tan solo puede resolver problemas linealmente separables, por ello se buscan formas de mejorarlo.

El **perceptrón multicapa** logra esta mejora mediante la conexión recurrente de perceptrones en línea, obteniendo varias capas de neuronas completamente conectadas entre si, siendo la salida de unas neuronas la entrada de otras. Sin embargo, para poder entrenar todas las neuronas distintas de la última, en las capas ocultas, se requiere un algoritmo de entrenamiento diferente, el algoritmo de backpropagation.

En el apéndice **II Introducción a las Redes Neuronales** se puede ver más en detalle y formalizado lo explicado hasta ahora, pero el objetivo es ver cómo se utiliza esta técnica en el problema planteado, el reconocimiento facial.

El perceptrón multicapa tiene muchas utilidades, y sirve de base para otros métodos basados en redes neuronales aún más sofisticados. La **aplicación general** de este método para el reconocimiento facial es la siguiente: Funcionan con una neurona de entrada para cada píxel y una de salida para cada clase. Aplica el algoritmo de retropropagación para hallar los pesos que minimizan el error, después se introduce la imagen de test y se obtendrá la clasificación de la salida.

Sin embargo, no se suele utilizar para este tipo de problemas, en general para cualquier problema que trate con imágenes. Esto se debe a varias razones:

- **Coste computacional:** Cada imagen está formada por miles de píxeles,

y como se ha mencionado, cada uno de ellos requiere una neurona, lo cual aumenta sustancialmente el tamaño necesario de la red. Esto provoca que si se busca un buen resultado, haya que emplear grandes cantidades de tiempo y recursos computacionales.

- **Estabilidad:** Al entrenar píxel a píxel, el perceptrón requiere que entre dos imágenes referentes a un mismo rostro, el mismo píxel tenga valores similares en ambas. Pero esto no tiene por qué ser así, ya sea por ligeras variaciones al tomar la foto en la posición, ángulo o iluminación.

De todos modos, cuando el número de clases no es muy elevado y la base de datos está controlada puede dar muy buen resultado. Sino, aumentando las neuronas y el número de etapas, también se pueden mejorar sus resultados, pero a costa de aún mayor coste computacional.

Las redes neuronales se pueden utilizar dentro de un sistema de reconocimiento facial como **clasificadores**, interviniendo tan solo al final del proceso, para las características obtenidas utilizando cualquiera de los métodos presentados previamente. Sin embargo, para esto es necesario una gran base de datos, ya que serán necesarias varias imágenes de un rostro para entrenar la red neuronal. De esta forma se reduce el número de datos de entrada y se vuelven menos sensibles a las variaciones en la imagen.

Además, se pueden usar de tal forma que sea la propia red neuronal la que selecciones cuáles son estas características más importante mediante los algoritmos de aprendizaje vistos previamente, por ejemplo para el número óptimo de componentes de la imagen en el método PCA, el uso de una red neuronal en la parte final del proceso permite particularizar este número para cada problema planteado. Sin embargo, esto puede llevar a un **sobreajuste**, por lo que se debe aplicar cuidadosamente para lograr la efectividad requerida (por ejemplo, un 90% de precisión) sin usar todas las componentes.

En el siguiente apartado, se continúa con este enfoque de aplicar las redes neuronales a datos de entrada filtrados para evitar estos problemas, en lo que se conocen como redes neuronales convolucionales.

3 Redes Neuronales de Convolución (CNN)

3.1 Convolución

Las CNN o redes neuronales convolucionales son un tipo de redes convolucionales profundas, que se utilizan comúnmente para el análisis visual de imágenes. El nombre de las CNN indica que emplea un tipo de operación matemática llamada convolución en algunas de sus capas, en lugar de las mul-

tiplicaciones matriciales habituales.

Una convolución es un tipo especial de operación lineal sobre una pequeña matriz o kernel. Además de la entrada, en cada convolución se utiliza un filtro distinto, cuyo valor más apropiado se estima utilizando el algoritmo de aprendizaje.

El objetivo es, mediante el uso de estos filtros, obtener diferentes características o patrones.

3.1.1 Formulación matemática de la operación de convolución

La convolución es una operación conmutativa entre dos funciones reales que depende de un parámetro t el cual se interpreta como el tiempo. Dependiendo de si t toma valores continuos o discretos, la convolución se define de las siguientes formas, respectivamente:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \quad o \quad \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

En el caso de las redes convolucionales, esta operación se define entre matrices o también llamadas tensores. El primer argumento, x, se conoce como entrada o imput, en este caso será la matriz de la imagen original de dimensiones $n \times m$. El segundo argumento, w, se conoce como kernel, filtro de convolución o mapa de características, y será otra matriz, cuadrada de menor tamaño $k \times k$ con k generalmente impar. Además, t hará referencia al píxel de la matriz en el cual nos situamos, es decir será un par de valores (i,j) dentro de los límites del tamaño de la imagen.

Con esto, la operación de convolución queda de la siguiente forma, con $p = \frac{k-1}{2}$

$$s(i,j) = (I * K)(i,j) = \sum_{m=-p}^{p} \sum_{n=-p}^{p} I(i,j)K(i-m, j-n)$$

Obsérvese, que por la simetría del kernel y la conmutatividad, también se puede expresar como:

$$s(i,j) = (K * I)(i,j) = \sum_{m=-p}^{p} \sum_{n=-p}^{p} K(i+m, j+n)I(i,j)$$

Esto permite visualizar mejor que el kernel recorre cada elemento de la imagen desde arriba a la izquierda hasta abajo a la derecha. Matricialmente, en la siguiente figura se puede apreciar cómo aplicar el filtro a la imagen no es más que situarse en un píxel, en este caso en el elemento (5,6) con valor

2, y aplicar el filtro a los píxeles de su alrededor, multiplicando uno a uno los valores y sumándolos todos para obtener el resultado final.

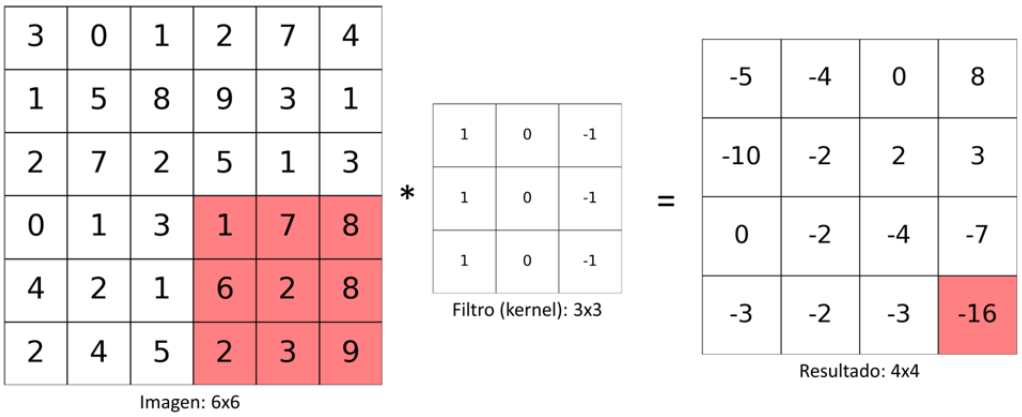


Figure 10: convolución [16]

3.2 Arquitectura de las CNN

Habitualmente se pueden encontrar 3 tipos de capas en la arquitectura CNN.

3.2.1 Capas convolucionales

En ellas se lleva a cabo la ya explicada operación de convolución hasta recorrer completamente la imagen de izquierda a derecha y de arriba a abajo. La metodología con la que se lleva a cabo esta convolución viene determinada por dos parámetros: el stride y el zero padding.

- El **stride** es cuánto se desplaza el filtro cada vez que se aplica. El stride base de 1 se puede apreciar en la imagen anterior, y proporciona una salida con mucha redundancia y poca reducción de la dimensionalidad, aunque esto también se puede lograr en otras capas posteriores. Mientras que, a medida que el stride se aproxima al tamaño del Kernel, obtendremos una salida de mucho menor tamaño y en la que cada píxel prácticamente solo sea utilizado para una operación de convolución. En cuanto a qué valor tomar, dependerá del problema en particular y de tal forma que que el volumen de salida sea un valor entero y no una fracción.
- **Zero padding:** Rellena el volumen de entrada con ceros alrededor del borde. Por lo que, tras el relleno correspondiente, y la aplicación del kernel, el tamaño original podría disminuir, aumentar o mantenerse igual, según lo que se busque.

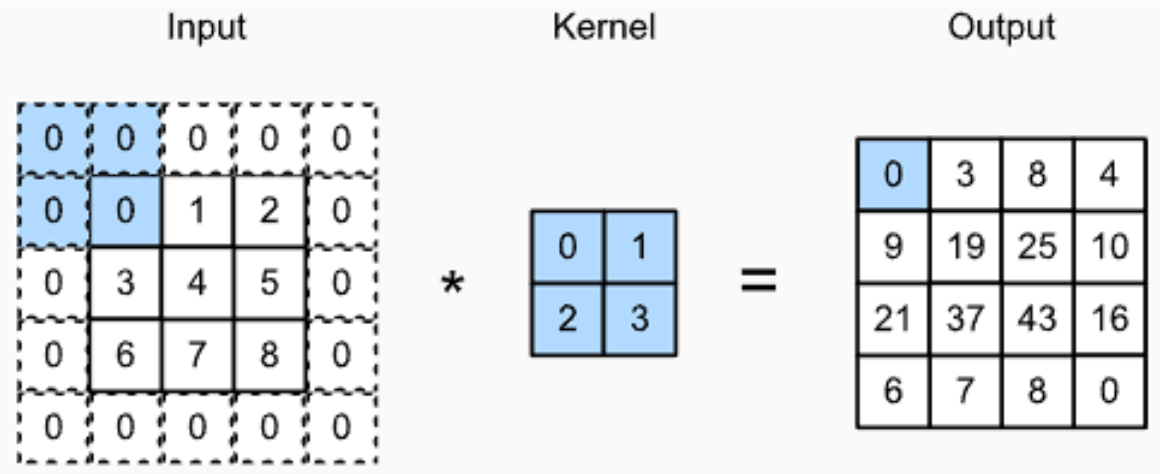


Figure 11: convolucion y zero padding [17]

3.2.2 Capas de Pooling

Esta capa también conocida como capa de agrupación es utilizada para **reducir el tamaño** de los mapas de características, lo cual proporciona tres ventajas: mayor eficiencia computacional, gracias a su menor tamaño de procesamiento; la mejor obtención de características dominantes, y una mayor robustez frente al ruido.

El procedimiento consiste en tomar un núcleo más pequeño que el kernel previo y aplicarle una operación simple como calcular su máximo o su media. Al igual que con la operación de convolución, variando el stride variará el tamaño de la matriz resultante.

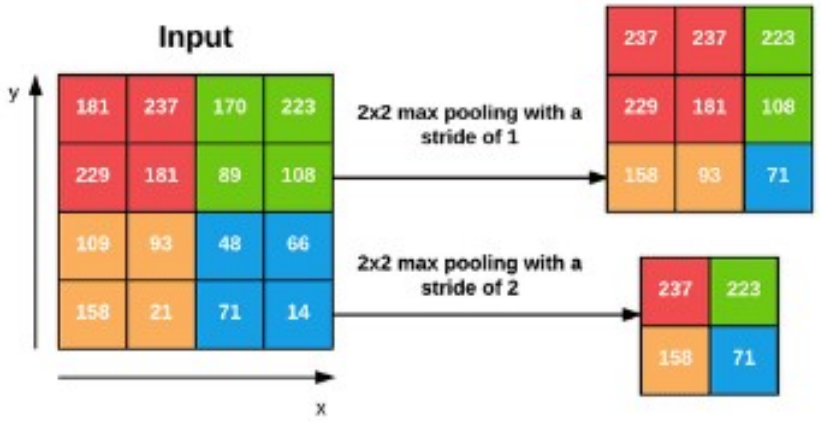


Figure 12: max pooling y stride [18]

3.2.3 Capas Inception

En general, para mejorar el resultado obtenido con una CNN, se debe aumentar su tamaño. Esto provoca dos problemas que se explican en la sección 3.6, un posible sobreajuste y un aumento del coste computacional, que de hecho será cuadrático debido a que todas las capas tienen sus neuronas conectadas con todas las de las capa anterior y posterior.

El objetivo de las capas Inception será reducir este coste computacional. Para ello, se aprovecha que, al aumentar el número de filtros, la capacidad añadida puede ser utilizada ineficientemente. Por ejemplo, cuando muchos de los pesos tengan valores cercanos a cero. Esto quiere decir que muchas de las interconexiones entre las capas no estarían siendo efectivas, por lo que sería beneficioso eliminarlas.

La arquitectura Inception funciona realizando un análisis al final de cada capa de la correlación de sus unidades, de tal forma que se agrupen las más correladas, y sean estas agrupaciones las que reciba la siguiente capa. Esto significaría que se acabaría con un gran número de grupos conectados en una sola región que se puede cubrir con convoluciones 1x1 en la siguiente capa, aunque, especialmente en las últimas capas, la dispersión puede ser mayor por lo que se requerirían convoluciones de mayor tamaño. Por conveniencia, los tamaños de los filtros que puede recibir una capa inception son los 1x1, 3x3 y 5x5.

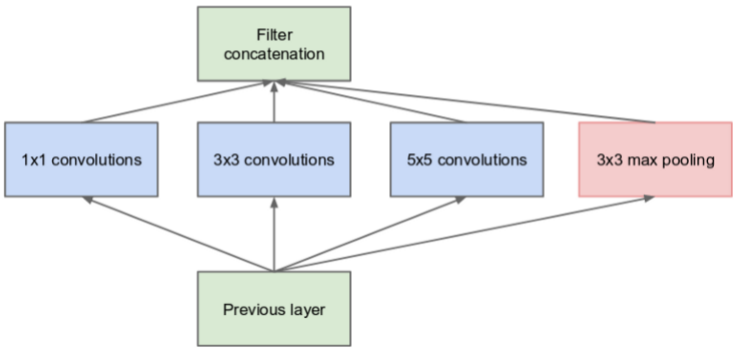


Figure 13: Módulo Inception, versión naïve [19]

Al ser las convoluciones 3x3 y 5x5 más costosas, se opta por una mejora de la arquitectura inception implementando convoluciones 1x1 previo a ellas, dando lugar a los módulos inception con reducción de la dimensionalidad.

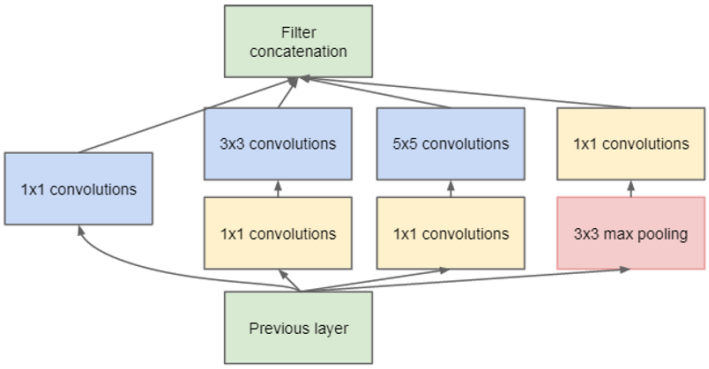


Figure 14: Módulo Inception con reducción de la dimensionalidad [19]

3.2.4 Capa completamente conectada

Suelen encontrarse al final de la red neuronal. Su nombre viene de que todas las neuronas están conectadas a todas las neuronas de la capa anterior, por lo que se recoge la información obtenida de todas las operaciones previas como un único vector y esta información se combina para identificar patrones más generales o para clasificar la salida.

De todas las capas de una red neuronal convolucional, estas son las más similares a las redes neuronales al uso, ya que su funcionamiento es muy similar al perceptrón multicapa explicado en **II Introducción a las Redes Neuronales**

3.3 Backpropagation

Tras pasar todas estas capas, la red puede emplear el algoritmo de backpropagation, que se puede ver explicado en detalle para un perceptrón multicapa en **II Introducción a las Redes Neuronales**. De forma resumida, consiste en volver a enviar la información a capas anteriores para recalcular los valores del kernel buscando mejorar el resultado obtenido. Esto puede dar muy buenos resultados en los datos de prueba pero no en los de test, esto se conoce como overfitting, que se explicará más adelante en 3.6.

3.4 Operaciones sobre valores de entrada a capas

3.4.1 Normalización

Durante el entrenamiento de las redes neuronales profundas cambia la distribución de las entradas de cada capa de acuerdo a los cambios que se producen en las capas anteriores. Esto produce varios **problemas en el proceso de entrenamiento:**

- **Ralentización del entrenamiento:** Se requiere una tasa de aprendizaje α más baja. Sino, pueden producirse cambios demasiado bruscos.
- **Inicialización:** Los parámetros no podrán tomar cualquier valor, habrá que inicializarlos más cuidadosamente para obtener resultados de forma más rápida.
- Dificultad para entrenar modelos con **saturaciones no lineales**.

A este fenómeno se le denomina como desplazamiento covariable interno, y para solucionarlo, se normalizan las capas de entrada. Esto es, convertir la entrada a una capa $\vec{x} \in \mathbb{R}^n$ a un vector de media 0 y desviación típica 1 mediante la operación:

$$norm(x_i) = \frac{x_i - \bar{x}}{\sigma_x} \quad \forall i = 1, ..., n$$

Donde $\bar{x}$ representa la media del vector y σ_x su desviación típica.

3.4.2 Softmax

La función softmax también llamada función exponencial normalizada aparece generalmente en las últimas capas ocultas. Se utiliza para proyectar los valores de la entrada al **intervalo [0,1]**. La operación para la entrada $\vec{x} \in \mathbb{R}^n$ es la siguiente:

$$softmax(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \quad \forall i = 1, ..., n$$

3.5 Feature Pyramid Networks (FPN)

Las redes piramidales de características consisten en la extracción de características en distintos niveles, por ello se denominan pirámide. Esto permite crear **varios mapas de características**, cada uno de ellos con mejor información que la imagen regular.

En un principio, se utilizaba la imagen completa en distintas resoluciones para detectar desde objetos grandes a pequeños. Sin embargo, esto consume mucha memoria y tiene un gran requerimiento computacional, por lo que se le da otro enfoque: a partir de la imagen original, crear mapas de características de distintos tamaños, cada vez con la información más comprimida, para obtener distintos niveles de información y poder acceder a cada uno de ellos. Sin embargo, en este proceso se puede haber perdido información para detectar objetos de menor tamaño, que en la pirámide de imágenes completas aparecería siempre, pero en los mapas de características se puede perder por ser pequeño relativo a otros objetos.

Su uso principal es en la segmentación y detección de objetos, ya que permite hallar primero objetos de gran tamaño en la imagen para después buscar otros más pequeños de ser necesario. Sin embargo, se les puede dar uso en reconocimiento, ya que permiten descartar un gran número de sujetos únicamente mediante una imagen en baja resolución, ateniéndose al pelo, tono de piel, tamaño de la cara u otras características de fácil identificación. Y más adelante, ir incluyendo las imágenes más detalladas tan solo en el subconjunto restante.

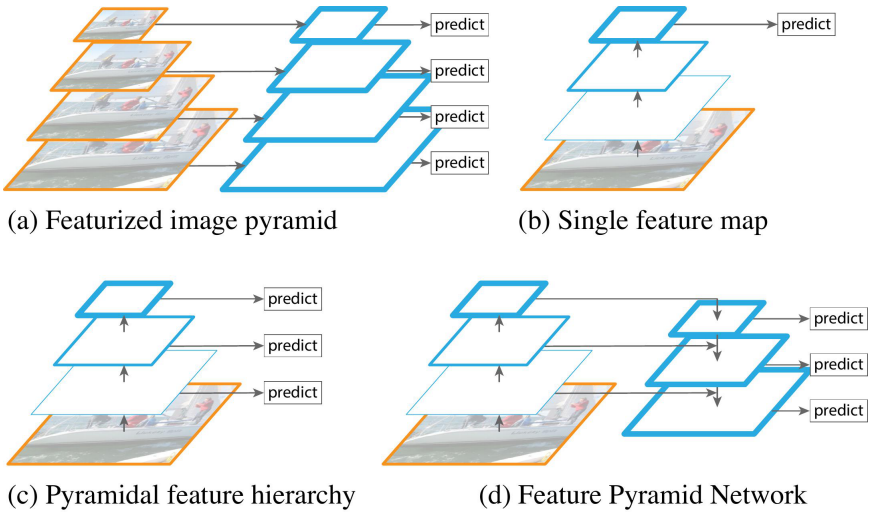


Figure 15: Funcionamiento de FPN [5]

3.6 Problemas y dificultades del uso de CNN

Un problema común en las redes neuronales es lo que se conoce como sobreajuste u **overfitting**, cuando se ajustan un número elevado de pesos en gran cantidad de neuronas, de tal forma que para las imágenes de entrenamiento se obtiene un clasificador perfecto, pero quizás no sea adecuado al introducir las imágenes de test.

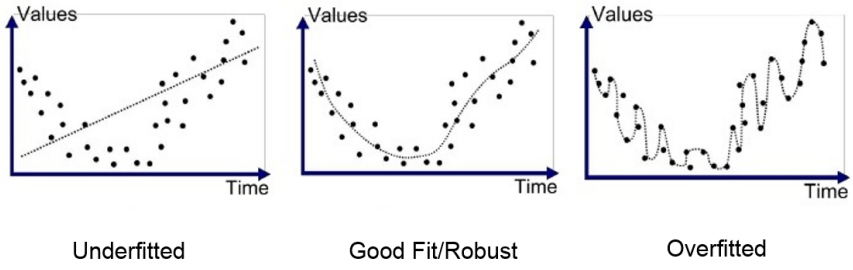


Figure 16: overfitting [12]

El método más común para evitar tener este problema es la disminución de peso (**weight decay**): esto es, agregar un término extra a la función de error de cada peso, por ejemplo en la regularización L_2 se añade $0.5\lambda w^2$.

Otra forma de eliminar el sobreajuste es eliminar neuronas aleatoriamente tras la salida de alguna capa, típicamente la de pooling (**dropout**).

Más allá de estas técnicas, también se pueden aplicar métodos propios de la ciencia de datos, como son:

- Dividir el conjunto de datos en: entrenamiento, validación y test.
- **Validación cruzada:** dividir el conjunto inicial en k subconjuntos y utilizar todos salvo uno para entrenamiento, y el último como validación. Repetir esto para todos los subconjuntos.
- Aumentar el tamaño de la base de datos, pero esto provocará mayor coste computacional.

Otro problema es el **coste computacional**, ya que, aunque se reduce sustancialmente respecto al uso de redes neuronales multicapa, sigue siendo muy elevado. Para ello, ya se mencionaron previamente las capas inception, y en la siguiente sección se presenta otra solución a este problema: las redes preentrenadas.

3.7 Redes preentrenadas

Anualmente se organiza la competición **ILSVRC** (ImageNet Large-Scale Visual Recognition Challenge), en la cual se deben clasificar grandes datasets de imágenes en ciertas clases. Los métodos que mayor efectividad están demostrando recientemente están basados en el uso de redes neuronales convolucionales. Algunas de las redes, las ganadoras o que demuestren gran efectividad, quedan entrenadas para reconocer todo ese conjunto de imágenes.

Es posible emplear una red pre-entrenada para clasificar imágenes, la cual ha sido previamente capacitada para identificar características relevantes y útiles en imágenes naturales, y utilizarla como base para aprender una tarea diferente. Algunas son accesibles desde el paquete DeepToolbox de Matlab, lo cual se usará en la sección de aplicación de las distintas técnicas, donde en lugar de una red CNN entrenada a mano, se usará **Alexnet**, que tiene las siguientes características:

- **Entrada:** Recibe imágenes cuadradas en formato RGB de tamaño 227x227x3
- **8 capas:** 5 convoluciones con sus capas de activación RELU así como max-pooling y normalización en algunas de ellas. En la imagen se pueden apreciar los tamaños y strides de cada una de estas capas.

- **Salida:** tras dos capas totalmente conectadas, se llega a la tercera capa totalmente conectada, en la cual hay 1000 neuronas, lo cual permite clasificar la imagen en 1000 clases distintas aplicando una función Softmax para ver qué neurona está más activa y asignar la imagen a esa clase.

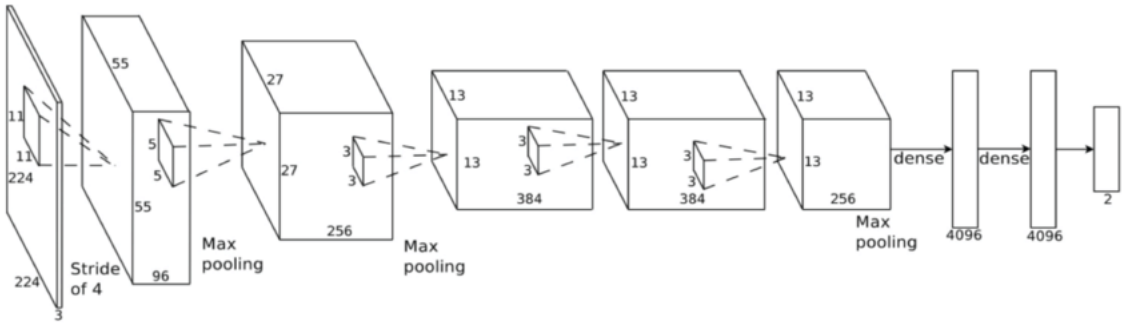


Figure 17: Estructura de Alexnet [13]

En la práctica, la información que se le tendrá que proporcionar a la red son el número de clases a clasificar, que deberá ser menor o igual a 1000, y para el resto se desactivarán neuronas, lo cual agilizará el proceso. Además, se puede variar:

- **El método de entrenamiento:** en las aplicaciones posteriores se usará el ya mencionado gradiente descendiente.
- **La velocidad de entrenamiento:** el ya mencionado valor α que sirve de potenciador o disminuidor del cambio en los pesos.
- **El máximo número de épocas o iteraciones:** Para reducir coste computacional y evitar overfitting. Además del número de datos de entrenamiento usados en cada una de las épocas (batch).

4 Aplicación de técnicas de reconocimiento facial

De todos los métodos vistos, realizaré una comprobación de la eficacia de los más prácticos para unos problemas que se presentarán a continuación. Los métodos basados en características locales han sido directamente descartados tanto por estar anticuados como por requerir un gran preproceso manual.

Acerca de las técnicas que si usaré, a continuación doy algunas indicaciones acerca de ciertos parámetros o implementaciones:

- **PCA:** se utilizará la media de las imágenes de entrenamiento para obtener los subespacios correspondientes.
- **LDA:** se implementará el algoritmo de Fisherfaces ya que la técnica original tiene un coste muy elevado, aprovechando el cálculo del subespacio utilizado para PCA.
- **Perceptrón multicapa:** el perceptrón empleado tiene 25 neuronas en la capa oculta con funciones tangencial-sigmoidales, el número de caras a reconocer (40 y 2) como neuronas de salida con función de activación máx para ver cuál está más activa, $\alpha = 0.001$ y pesos iniciales aleatorios.
- **PCA+perceptrón:** con la mitad de los datos de entrenamiento de cada cara se genera el subespacio en el que se representarán las imágenes. Después, con el resto de datos de entrenamiento se entrena un perceptrón multicapa como el anterior, que en lugar de recibir vectores de píxeles como entrada, recibe las características de las imágenes en el subespacio anterior, de mucho menor tamaño que los píxeles.
- **CNN:** usaré la red preentrenada de Alexnet con el algoritmo del gradiente descendente, $\alpha = 0.001$, 20 épocas y mínimo 64 imágenes de entrenamiento por época.

Por lo tanto, la comparación se dará entre PCA, LDA, un perceptrón multicapa y una red convolucional preentrenada.

4.1 Problemas planteados

4.1.1 Primer problema

Bajo una base de datos **controlada**, con ligeras variaciones en la pose o expresiones faciales, pero con un gran número de sujetos y pocas muestras de cada uno, la idea es comprobar qué métodos son mejores a la hora de separar sujetos cuando muchos de ellos pueden tener características comunes, pero con fotos similares entre si para cada uno.

La base de datos empleada es **ORL**, obtenida de [20]. Está formada por 40 sujetos, de los cuales se tienen 10 imágenes diferentes en distintas poses. Usaré las 9 primeras imágenes para el entrenamiento, y la última para el test. A continuación se pueden ver las 9 imágenes de entrenamiento del primer sujeto.

Figure 18: Base de datos primer problema

4.1.2 Segundo problema

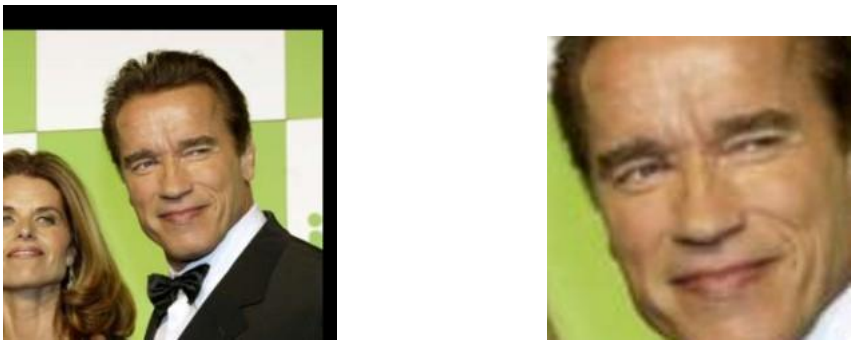
Bajo bases de datos **no controladas**, comprobar qué método ofrece mejores resultados. En esta ocasión, la idea es tener pocos sujetos y muchas imágenes para cada uno, con la dificultad de que estas imágenes serán muy variables entre si. Ahora el objetivo no es ver qué método separa mejor los conjuntos, sino cuál identifica mejor las características de cada uno de ellos.

La base de datos es **LFW (Labeled Faces in the Wild)**, ha sido obtenida de [21]. La base de datos cuenta con imágenes de famosos obtenidas a lo largo del tiempo en situaciones naturales, no controladas. En todas ellas se les ve el rostro pero no está centrado. Como muchos de los rostros cuentan con muy pocas imágenes, primero hice una selección de aquellos con más de 20 rostros reconocibles, quedando tan solo 4, los correspondientes a: Arnold Schwarzenegger, Gloria Macapagal Arroyo, Luiz Inacio Lula da Silva y Megawati Sukarnoputri.

Los 20 primeros rostros se utilizarán para el **entrenamiento**, y los restantes para el test. En las imágenes de test no habrá el mismo número de imágenes correspondientes a cada rostro, pero lo importante es que en el entrenamiento las haya para no sesgar el resultado.

Después, para centrar los rostros, se utilizaron técnicas de **detección de rostros**, en particular la función incoporada en matlab vision.CascadeObjectDetector que utiliza el algoritmo de Viola-Jones explicado más en detalle en la sección

II Introducción a las Redes Neuronales. En la siguiente figura se aprecia el efecto de aplicar esta técnica a una de las imágenes seleccionadas.



(a) Imagen previa a detección (b) Rostro detectado en la imagen

Figure 19: Imagen de Arnold Schwarzenegger antes y después de detectar el rostro

4.2 Resultados y comparación

Los resultados se presentan a modo de tabla midiendo tanto la precisión como el tiempo de ejecución del método.

4.2.1 Primer problema

Método	**Precisión**	**Tiempo de ejecución** (en segundos)
PCA	0.7	3.23
Fisherfaces	0.7	4.80
Perceptrón	0.125	172.81
PCA+Perceptrón	0.8	14.29
AlexNet(CNN)	0.975	191.46

- **Tiempo de ejecución:** Como era de esperar, en la tabla se observa que las técnicas basadas en inteligencia artificial son bastante más lentas que las otras, especialmente cuando se utiliza la imagen al completo, esto podría llegar a ser un problema si la base de datos fuera mucho más grande. Aún así, en todas las técnicas la clasificación siempre es un proceso muy rápido, por lo que una vez entrenadas es indiferente utilizar una u otra.

- **Precisión:** El mejor resultado se obtiene con las dos técnicas más avanzadas de inteligencia artificial presentadas. En particular, la red preentrenada solo comete un error. Sin embargo, como era de esperar, el

perceptrón por si solo no da buen resultado, ya que trabajar con los píxeles de la imagen en lugar de aplicar previamente filtros u operaciones no tiene sentido. La técnica de PCA también ha dado muy buen resultado, funcionando de forma parecida con o sin perceptrón como clasificador, y también a LDA, la cual no lo mejora probablemente a que como se explicó en la sección 2.2.5, el subespacio generado por PCA no tiene por qué ser el mejor para LDA.

4.2.2 Segundo problema

Método	**Precisión**	**Tiempo de ejecución** (en segundos)
PCA	0.5714	0.64
Fisherfaces	0.5714	0.40
Perceptrón	0.2857	36.96
PCA+Perceptrón	0.5714	0.85
AlexNet(CNN)	0.9286	58.93

- **Tiempo de ejecución:** Los resultados son equivalentes al primer problema pero con tiempos mucho menores debido al menor número de imágenes a tratar. Aún así, se puede sacar una nueva conclusión, y es que todos los tiempos son del orden de una cuarta parte de los del primer problema salvo para el perceptrón con PCA. Esto hace sospechar que esta técnica es más sensible que otras a la variación de tamaño en la base de datos, lo cual podría ser un problema al tratar de usarla para resolver un problema mucho mayor.

- **Precisión:** En este caso el mejor resultado se da indiscutiblemente con la red neuronal neuronal convolucional preentrenada, lo cual la sitúa como la más eficaz para el reconocimiento facial. En este caso la diferencia entre PCA con y sin perceptrón como clasificador se han reducido hasta dar el mismo resultado, y en ambos casos mucho peor que para el anterior problema, lo cual indica que para bases de datos no controladas la técnica de PCA no es tan buena.

Los resultados son muy similares al primer problema, aunque todos los métodos han perdido precisión al tratarse de una base de datos peor salvo el perceptrón. Esto se debe a que la mejor función del perceptrón es como separador, y hay un mayor número de imágenes para entrenar y menos categorías, lo cual ayuda a encontrar una separación más fácil.

5 Conclusiones y futuras líneas de investigación

En este trabajo se han explicado diferentes técnicas usadas para el reconocimiento facial, siguiendo su evolución desde técnicas muy intuitivas pero manuales hasta técnicas más abstractas y la incorporación de la inteligencia artificial. Sobre esto último, se ha profundizado en el funcionamiento interno de las redes neuronales convolucionales (CNN).

Además, en el último apartado se ha puesto en práctica lo explicado a lo largo del trabajo, programando las distintas técnicas en matlab y aplicándose a varios problemas para comprobar que las redes neuronales convolucionales utilizadas para el reconocimiento facial presentan una amplia superioridad en cuanto a la precisión respecto a las demás técnicas, que funcionaron particularmente mal en el segundo problema, para el cual la base de datos no estaba controlada, lo cual como ya se mencionó en la introducción, ha sido el gran problema de estas técnicas a lo largo de la historia.

Estas mejoras mediante el uso de la inteligencia artificial han permitido que a día de hoy el reconocimiento facial se pueda dar hasta en la vida cotidiana, ya que está incorporado en los móviles, sistemas de alarma de hogares, coches, etc. Es por ello que se ha reducido la necesidad de investigación en este campo en cuanto a la eficacia, y el objetivo ahora es mejorar la velocidad de ejecución.

En la mayoría de ámbitos esto no es una necesidad imperiosa, sin embargo, hay otras ramas de la visión por ordenador en las cuales todavía no se ha alcanzado el punto necesario de velocidad más precisión requerido. Esto pasa por ejemplo en la visión artificial, a la cual se hizo referencia en la introducción, y en la cual la velocidad de reacción es primordial, ya que por ejemplo es lo que se utiliza para desarrollar la conducción autónoma. Si bien en ese campo los rostros no son una prioridad para las aplicaciones buscadas, son un buen método de entrenamiento para mejorar estas técnicas.

Por otro lado, se sigue trabajando en el reconocimiento facial en 3 dimensiones, mejorando las técnicas de adquisición de la imagen y ampliando las bases de datos, que también tienen aplicación en videojuegos o series y películas animadas en 3D, a la hora de diseñar rostros o asemejarse lo máximo posible a un rostro ya existente.

Bibliografía

Referencias

[1] M. Anggo and L. Arapu, *Face recognition using fisherface method*, Journal of Physics: Conference Series, 1028 (2018), p. 012119. (Citado en página 19.)

[2] E. Cabello Pardos, *Técnicas de reconocimiento facial mediante redes neuronales*, PhD thesis, Informatica, 2004.

[3] N. Casado Beinat, *Redes neuronales convolucionales y aplicaciones*, (2022).

[4] A. B. M. Díaz, *Reconocimiento facial automático mediante técnicas de visión tridimensional*, PhD thesis, Universidad Politécnica de Madrid, 2004. (Citado en página 20.)

[5] H. Gao and J. W. Davis, *Why direct lda is not equivalent to lda*, Pattern Recognition, 39 (2006), pp. 1002–1006.

[6] M. Guerrero Moñús, *Aplicación de técnicas de aprendizaje profundo en imágenes para el reconocimiento de objetos*, (2022).

[7] C. Justo de Frías, *Visión artificial aplicada en la identificación de objetos y su parametrización geométrica*, B.S. thesis, 2019.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *Imagenet classification with deep convolutional neural networks*, Communications of the ACM, 60 (2017), pp. 84–90.

[9] K.-C. Kwak and W. Pedrycz, *Face recognition using a fuzzy fisherface classifier*, Pattern Recognition, 38 (2005), pp. 1717–1732. (Citado en página 19.)

[10] P. Larranaga, I. Inza, and A. Moujahid, *Tema 8. redes neuronales*, Redes Neuronales, U. del P. Vasco, 12 (1997), p. 17. (Citado en página 47.)

[11] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, *Feature pyramid networks for object detection*, in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 2117–2125.

[12] C. P. J. H. . P. E. B. Martinsanz, G. P., *Aprendizaje profundo*, RC Libros, 2021.

[13] A. Pedraza, J. Gallego, S. Lopez, L. Gonzalez, A. Laurinavicius, and G. Bueno, *Glomerulus classification with convolutional neural networks*, 06 2017, pp. 839–849. (Citado en página 31.)

[14] L. B. Pérez, *Reconocimiento facial basado en puntos característicos de la cara en entornos no controlados*, Universidad Autónioma de Madrid (España)-Departamento de Ingeniería Informática, (2013), p. 17. (Citado en página 9.)

[15] M. Pérez Rodríguez et al., *Reconocimiento eficiente de caras mediante deep learning a partir de imágenes en el espectro visible*, (2021).

[16] J. M. Sierra Ramos, *Introducción a las redes neuronales artificiales*, (2022).

[17] E. Temprano Coleto, *Métodos de aprendizaje profundo para la segmentación semántica de personas*, (2020).

[18] Y.-Q. Wang, *An analysis of the viola-jones face detection algorithm*, Image Processing On Line, 4 (2014), pp. 128–148. (Citado en páginas 42 y 44.)

[19] H. Xiong, M. Swamy, and M. Ahmad, *Two-dimensional fld for face recognition*, Pattern Recognition, 38 (2005), pp. 1121–1124.

Webgrafía (consultada por última vez en Junio de 2023)

[1] BeeDigital. Historia y evolución del reconocimiento facial[Online]
https://www.beedigital.es/tendencias-digitales/historia-y-evolucion-del-reconocimiento-facial/

[2] Nuruzzaman Faruqui. Face recognition using Matlab[Online]
https://www.nzfaruqui.com/face-recognition-using-matlab/

[3] Sanketh Kini. Getting Started With CNN[Online]
https://medium.com/@kinisanketh/getting-started-with-cnn-18c03efc7d06

[4] Websa100. 10 usos posibles del reconocimiento facial que ya son una realidad[Online]
https://www.seoptimer.com/es/blog/usos-reconocimiento-facial/

[5] Jonathan Hui. Understanding Feature Pyramid Networks for object detection (FPN)[Online]
https://jonathan-hui.medium.com/understanding-feature-pyram

id-networks-for-object-detection-fpn-45b227b9106c (Citado en página 29.)

[6] Iberdrola. ¿Qué es la visión artificial y cuáles son sus aplicaciones?[Online] https://www.iberdrola.com/innovacion/vision-artificial

[7] Keynua. Identificación a través de biometría facial: 2D vs 3D[Online] https://es.linkedin.com/pulse/identificaci%C3%B3n-trav%C3%A9s-de-biometr%C3%ADa-facial-2d-vs-3d-keynua (Citado en página 2.)

[8] Abraham Rojas Castro. SISTEMA BIOMETRICO[Online] https://example87113.wordpress.com/2016/05/09/sistema-biometrico/

[9] Scholarpedia. Elastic Bunch Graph Matching[Online] http://www.scholarpedia.org/article/Elastic_Bunch_Graph_Matching (Citado en página 11.)

[10] Matlab Mathworks: Redes Neuronales Profundas preentrenadas[Online] https://es.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html

[11] Rohan Gupta. Breaking Down Facial Recognition: The Viola-Jones Algorithm[Online] https://towardsdatascience.com/the-intuition-behind-facial-detection-the-viola-jones-algorithm-29d9106b6999 (Citado en páginas 42 y 44.)

[12] Anup Bhande. What is underfitting and overfitting in machine learning and how to deal with it.[Online] https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76 (Citado en página 29.)

[13] Yashwant Saini. Face Recognition using Fisherfaces.[Online] https://iq.opengenus.org/face-recognition-using-fisherfaces/ (Citado en página 5.)

[14] Datysoc[Online] https://twitter.com/datysoc/status/1505931572859617289 (Citado en página 9.)

[15] Jun de WU. Análisis de Componentes Principales: Una breve introducción matemática[Online] https://blog.damavis.com/analisis-de-componentes-principales-una-breve-introduccion-matematica/ (Citado en página 16.)

[16] Miguel Sotaquirá. La Convolución en las Redes Convolucionales[Online] https://www.codificandobits.com/blog/convolucion-redes-convolucionales/ (Citado en página 24.)

[17] washington.edu. Convolutional Neural Networks[Online] https://courses.cs.washington.edu/courses/cse446/22wi/sections/08/convolutional_networks.html (Citado en página 25.)

[18] Adrian Rosebrock. Convolutional Neural Networks (CNNs) and Layer Types[Online] https://pyimagesearch.com/2021/05/14/convolutional-neural-networks-cnns-and-layer-types/ (Citado en página 25.)

[19] Hongsuk's Note. [CNN] GoogLeNet[Online] https://hnsuk.tistory.com/32 (Citado en páginas 26 y 27.)

[20] Base de Datos ORL[Online] https://paperswithcode.com/dataset/orl (Citado en página 33.)

[21] Base de Datos Labeled Faces in the Wild(LFW)[Online] http://vis-www.cs.umass.edu/lfw/ (Citado en páginas 33 y 49.)

[22] Alejandro Cartas. Diagrama de un perceptrón con cinco señales de entrada.Wikipedia[Online] https://commons.wikimedia.org/wiki/File:Perceptr%C3%B3n_5_unidades.svg#/media/Archivo:Perceptrón_5_unidades.svg (Citado en página 45.)

Videografía

[1] Steve Brunton. Principal Components Analysis(PCA), 2020 https://www.youtube.com/watch?v=fkf4IBRSeEc

[2] Nuruzzaman Faruqui, MATLAB Face Detection, Tracking and Recognition playlist of videos https://www.youtube.com/playlist?list=PL9be9JpeQ7IOeLn23wtCttKYM7XIQ6-ow

[3] Dr. Rashi Agarwal. PCA for Face Recognition- Part III, 2017 https://www.youtube.com/watch?v=158XPhPiZLc

[4] Hackeando Tec. Curso de Redes Neuronales Artificiales playlist of videos, 2015 https://www.youtube.com/playlist?list=PLIyIZGa1sAZo_eY8PpuTxfLsja_iyytSE

[5] Aprende e Ingenia. IDENTIFICACION Y RECONOCIMIENTO FACIAL EN TIEMPO REAL | [MSER Matlab], 2020
https://www.youtube.com/watch?v=EiNAgBnH0eM

Apéndices

I Algoritmo de Viola-Jones para la detección de objetos

Este algoritmo fue desarrollado en 2001 por Paul Viola y Michael Jones. Pese a su antigüedad, sigue siendo uno de los algoritmos más potentes para la detección de rostros, especialmente cuando se busca **rapidez y bajo coste computacional**. El algoritmo consta de dos etapas: entrenamiento y detección.

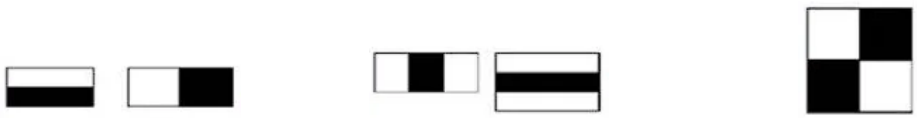
I.I Detección

El algoritmo trabaja con imágenes frontales en escala de grises. Si la imagen es a color, la detectará en blanco y negro y después se trasladará el resultado a la imagen original.

El procedimiento consiste en comprobar si hay un rostro en una **bounding box**, un rectángulo de cierto tamaño que se ha fijado previamente. Este rectángulo se desplaza a lo largo de la imagen y el algoritmo detecta cuándo hay características propias de un rostro contenidas en él. A estas se les conoce como características Haar (Haar-Like features). Una vez se ha recorrido toda la imagen, con los datos obtenidos por todos los rectángulos se determina dónde se encuentra el rostro.

I.I.I Haar-Like features

Están basadas en las ondículas de Haar, de ahí su nombre. Buscan diferencias en los tonos de blanco y negro de la imagen propios de elementos de un rostro como las cejas o la nariz. Para el algoritmo, interesan en particular 3 tipos de características Haar.



(a) De borde (b) De línea (c) De cuatro lados

Figure 20: Tipos de características Haar [11]

Por ejemplo, las de borde horizontales y de línea verticales sirven para detectar las cejas y la nariz, respectivamente.

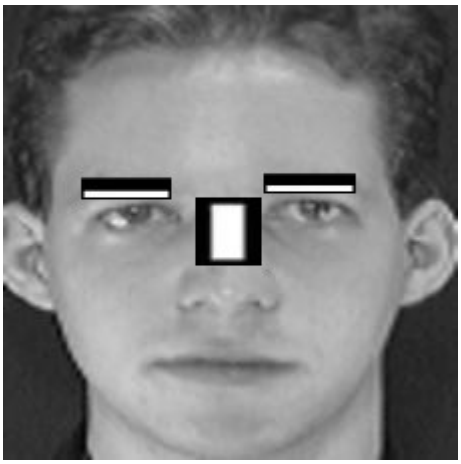


Figure 21: Cejas y nariz con Haar

Cada una de estas características se obtiene de hacer una **operación** distinta sobre la matriz que representa la escala de grises de los píxeles de la imagen. Por ejemplo, para una característica de borde horizontal, se realizarán sumas por filas y después se buscará dónde la diferencia entre estas sumas es elevada, que indicará que habrá un gran salto en la luminosidad. Para un patrón general P, que valdrá 0 donde sea blanco y 1 donde sea negro, en una imagen I en escala de grises, ambos de misma dimensión $N \times N$ la operación sería:

$$\sum_{i=1}^{N}\sum_{j=1}^{N} I(i,j)1_{P(i,j)=0} - \sum_{i=1}^{N}\sum_{j=1}^{N} I(i,j)1_{P(i,j)=1}$$

Pero esto requiere muchos cálculos, así que se utiliza la imagen integral.

I.I.II Imagen Integral

Se denomina así a la imagen producida por la suma de todos los valores de luminosidad de los píxeles arriba a la izquierda de uno mismo, es decir, el **valor acumulado**. A partir de la imagen I, su imagen integral II se obtiene como:

$$II(i,j) = \sum_{s=1}^{i}\sum_{t=1}^{j} I(s,t)$$

Esto ayuda a que los cálculos sean mucho más rápidos ya que el algoritmo funciona con las bounding boxes como ya se explicó. Así, si se quiere saber la suma de una fila de la bounding box, solo hay que restar el primer valor al último. Y lo mismo para el resto de características.

I.II Entrenamiento

El entrenamiento es mediante **aprendizaje supervisado:** se le deben dar tanto imágenes etiquetadas con rostros como sin ellos, para que no solo aprenda a encontrarlos sino a saber cuándo no hay. El método de aprendizaje que se utiliza es boosting adaptativo (AdaBoost).

I.II.I AdaBoost

El boosting adaptativo (AdaBoost) consiste en que la clasificación de las imágenes se hace utilizando las características Haar secuencialmente hasta formar un clasificador fuerte, que será una combinación de ellas ponderadas por pesos.

- **1)** Se prueban todas las características para ver cuál clasifica mejor las imágenes de entrenamiento, y se utiliza esa como característica principal, dándole un mayor peso.
- **2)** Para las imágenes mal clasificadas, se busca la característica que mejor funciona en ellas. No tiene por qué ser la segunda mejor para el conjunto completo, sino la que mejor se complementa con la ya seleccionada. El peso de esta característica deberá ser menor que el de la anterior, que era más importante, pues sino se pueden producir errores en las imágenes que ya se habían clasificado correctamente.
- **3)** Se repite este proceso hasta clasificarlas todas correctamente, o un porcentaje muy elevado en caso de querer evitar un posible sobreajuste o reducir el coste computacional.

Para optimizar este proceso, se introduce el concepto de **cascada**. Consiste en hacer el proceso por pequeñas subventanas. En ellas se busca que esté la característica más importante. En caso de no estar, se descartará la subventana, y si si está, se buscará la siguiente característica más importante y de nuevo se descartará si no está. Así hasta el número que se considere importante. De esta forma se reduce la cantidad de operaciones a realizar ya que no se calculan todas las características para cada subventana porque muchas son previamente descartadas. De aquí viene el nombre de la función de matlab vision.CascadeObjectDetector. Se puede ver más acerca del algoritmo en [11].

II Introducción a las Redes Neuronales

II.I Perceptrón

El **perceptrón** es la unidad básica de una red neuronal. Su estructura trata de imitar el funcionamiento de una neurona del cerebro humano, y es la siguiente: el perceptrón tiene n entradas $\{x_1, ..., x_n\}$ con unos pesos asignados $\{w_1, ..., w_n\}$ y un umbral w_0, que se usarán para proporcionar una única salida y.

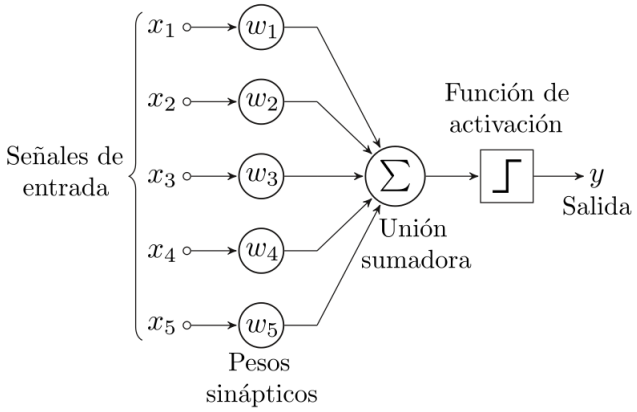


Figure 22: Estructura de un perceptrón [22]

II.II Funciones de activación

La salida mencionada previamente se calcula mediante una función de activación $y = f(x) = f(wo + \sum_{i=1}^{n} x_i w_i)$. Esta será una función no lineal, ya que, en caso contrario, el método sería equivalente a una regresión lineal. A continuación se presentan varios ejemplos de estas funciones y en qué redes se utilizan:

- **Función de salto binaria** esta función devolverá 1 si la neurona está activa $y = f(x) \geq 0$, y 0 sino. Esta función se utiliza en los perceptrones más básicos, con objetivos sencillos, por ejemplo para clasificar un conjunto linealmente separable.
- **Función sigmoidal** $f(x) = \frac{1}{1+e^{-x}}$ permite obtener un gran rango de valores para saber no solo si la neurona está activa o no, sino el grado en que lo está, penalizando los valores que se encuentran cerca del 0 (los más dudosos de clasificar). Pero su derivada no es computacionalmente eficiente, por ello no se usará en estas redes.

- **Función tangencial sigmodial o hiperbólica** $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ forma similar a la anterior, y con derivada más eficiente. Por ello es la que utilizaré en las capas ocultas del perceptrón multicapa.

- **Función RELU** $f(x) = max(0, x)$ devuelve solo valores positivos, aunque puede dar el problema de no estar acotada superiormente. Su gran ventaja es la reducción de coste computacional por la sencillez de calcular tanto su valor como su derivada (0 o 1 dependiendo de en qué tramo se encuentre el valor de activación x)

II.III Método de optimización: gradiente descendiente

Los pesos se pueden asignar manualmente, pero la potencia de las redes neuronales reside en poder recalcularlos buscando mejorar el resultado en cada iteración.

El método que se emplea para esto es el gradiente descendente, el cual es un método de **aprendizaje supervisado**: se conoce la variable respuesta, es decir, el valor que sabemos que tomará y. Esto permite calcular el error cometido con esos pesos. El procedimiento entonces será desplazar cada error en dirección opuesta al gradiente de la función f en $\{x_1, ..., x_n\}$.

Para **inicializar** el algoritmo, se parte de unos pesos iniciales que pueden ser aleatorios o designados por el propio usuario, junto a un valor α que servirá de **potenciador o disminuidor** del cambio que se producirá en los pesos: un valor muy elevado provocará saltos muy grandes, lo cual puede provocar saltarse el óptimo, mientras que un valor pequeño hará que los pesos varíen lentamente hacia el óptimo, pero siguiendo la dirección correcta. Por ello es importante elegir correctamente este valor.

Así, en cada iteración se recalcularán los pesos de la siguiente forma:

$$w(t+1) = w(t) - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(wo + \sum_{i=1}^{n} x_i w_i)$$

Sin embargo, un perceptrón por si solo tan solo puede resolver problemas linealmente separables, independientemente de lo bueno que sea el entrenamiento y la cantidad de iteraciones usadas, es por eso que se trabaja con el perceptrón multicapa, que presenta una mejora sustancial y es capaz de resolver problemas más complejos.

II.IV Perceptrón multicapa

La estructura del perceptrón multicapa se basa, como su nombre indica, en conectar varias capas de neuronas, es decir, varios **perceptrones en línea**, de tal forma que la capa que recibe los datos recibe el nombre de capa de entrada,

la que da los resultados de la red capa de salida y la capa o capas intermedias entre ambas se llaman **capas ocultas**.

El perceptrón multicapa está **completamente conectado**: existen conexiones entre todas las neuronas de una capa y todas las neuronas de la siguiente capa, y los pesos ahora se asignarán no solo a las variables de entrada, sino a cada una de estas conexiones. Si utilizamos los algoritmos de entrenamiento mencionados anteriormente, no obtendríamos ninguna mejora respecto a un perceptrón normal, ya que tan solo se entrenarían los pesos de la capa de salida. Por eso se propone el algoritmo de backpropagation, que explicaré a continuación.

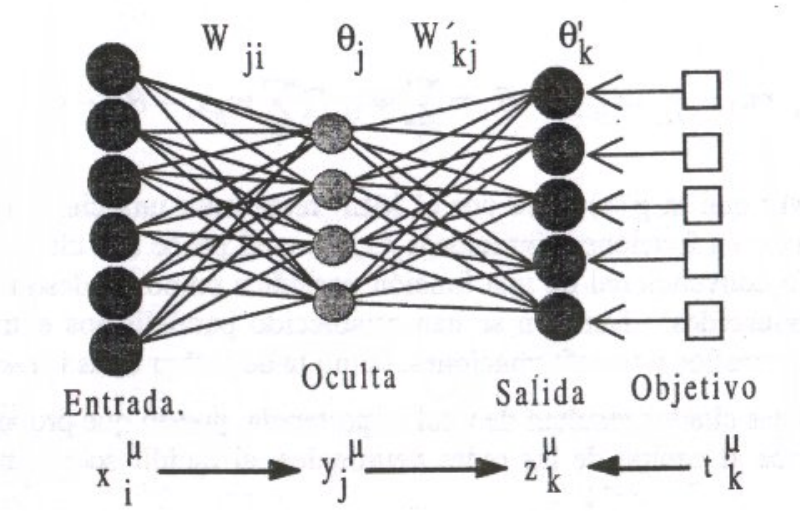


Figure 23: Estructura de un perceptrón multicapa [10]

II.IV.I Algoritmo de backpropagation

El algoritmo es una generalización del algoritmo de entrenamiento de un perceptrón simple:

Se parte del patrón de entrenamiento p-ésimo formado por una entrada y una salida deseada $\vec{x_p} \in \mathbb{R}^n$, $\vec{d_p} \in \mathbb{R}^m \quad \forall p \in \{1, ..., P\}$. A este también se corresponden las salidas obtenidas en cada neurona i de la capa l y_{ip}^l (definiendo $\vec{x_p} = \vec{y_p^0}$), que se corresponden a la entrada de la capa l+1, hasta la última capa L, en la cual se obtiene la salida de la red.

Con esta salida final se calcula el **error** para cada patrón:

$$E_p = \frac{1}{2} \sum_{i=1}^{m} (d_{ip} - y_{ip}^L)^2$$

que es lo que se buscará minimizar variando los pesos w_{ji}^l que representan la conexión que va de la j-ésima neurona en la capa l-1 a la i-ésima neurona en la capa l. Utilizamos la expresión del error en función de estos pesos, esto es, sustituir:

$$y_{ip}^L = f^L(\sum_{j=1}^{n} w_{ji}^L y_{ip}^{L-1} + \theta^L) = f^L(Neta_{ip}^{L-1}(w_{1i}^L, ..., w_{ni}^L, \theta^L))$$

Lo que se busca es hacer tender a 0 la derivada de E_p respecto de cada uno de los pesos w_{ji}^L, para ello se aplica la regla de la cadena :

$$\frac{\partial E_p}{\partial w_{ji}^L} = \frac{\partial E_p}{\partial f^L}\frac{\partial f^L}{\partial Neta_{ip}^{L-1}}\frac{\partial Neta_{ip}^{L-1}}{\partial w_{ji}^L} = -(d_{ip} - y_{ip}^L)(f^L)' y_{jp}^{L-1} := \delta_i^L y_{jp}^{L-1}$$

Y en particular $\frac{\partial E_p}{\partial \theta^L} = \delta_i^L$, donde δ_i^L lo definimos como el **error imputado a la neurona i-ésima de la capa L**.

Este proceso se repite de forma análoga para la capa L-1, obteniendo:

$$\frac{\partial E_p}{\partial w_{kj}^{L-1}} = \delta_i^L w_{ji}^L (f^{L-1})' y_{kp}^{L-2} := \delta_j^{L-1} y_{kp}^{L-2} \quad y \quad \frac{\partial E_p}{\partial \theta^{L-1}} = \delta_j^{L-1}$$

Y de forma iterativa $\frac{\partial E_p}{\partial w_{kj}^{l-1}} = \delta_j^{l-1} y_{kp}^{l-2}$ y $\frac{\partial E_p}{\partial \theta^{l-1}} = \delta_j^{l-1} \quad \forall l = 2, ..., L-1$ capas ocultas con $\delta_j^{l-1} = \delta_i^l w_{ji}^l (f^{l-1})'$.

La derivada de f dependerá de la función utilizada, que será generalmente una **función sigmoidal** en las capas ocultas: $f'(Neta_{ip}^l) = \frac{e^{Neta_{ip}^l}}{(1+e^{Neta_{ip}^l})^2}$, o una **función tangencial hiperbólica** $f'(Neta_{ip}^l) = \frac{1}{(e^{Neta_{ip}^l} + e^{-Neta_{ip}^l})^2}$. En la capa de salida, se suele utilizar una función lineal, con derivada igual a 1.

Finalmente, se actualiza el vector de pesos de forma equivalente a como se hizo en el algoritmo del gradiente:

$$w_{ji}^l = w_{ji}^l - \alpha \frac{\partial E_p}{\partial w_{ji}^L} \quad \forall i, j = 1, ..., n; \quad \forall l = 1, ..., L$$

,

III Códigos empleados

Todos los códigos están en lenguaje Matlab. Se asume que tanto códigos como carpetas con imágenes están en la misma carpeta, sino, habrá que fijar el directorio separado en el que se encuentren las imágenes al inicio del código.

III.I Códigos auxiliares

Seleción de rostros con suficiente número de imágenes del conjunto de datos [21]

```
clc
clear all
close all
warning off
f ='lfw';
g = 'Carpeta_Destino';
d=ls(f);
d(1,:)=[];
d(1,:)=[];
for i=1:size(d)
    ff = fullfile(f,d(i,:));
    dd=ls(ff);
    dd(1,:)=[];
    dd(1,:)=[];
    if size(dd)>28
        mkdir(fullfile(g,d(i,:)))
        a = strsplit(d(i,:));
        for j=1:size(dd)
            b = strsplit(dd(j,:));
            I=imread(fullfile(f,a{1,1},b{1,1}));
            imwrite(I,fullfile(g,a{1,1},b{1,1}));
        end
    end
end
```

Conversión de imágenes a tamaño 227×227, necesario para CNN, además utilizado para separar imágenes de entrenamiento de test.

```
clc
clear all
close all
warning off
f ='Carpeta_Origen';
s = 'Carpeta_Destino_Entrenamiento';
t = 'Carpeta_Destino_Test';
ntrain = 9;

d=ls(f);
d(1,:)=[];
d(1,:)=[];
for i=1:size(d)
        a = strsplit(d(i,:));
        ff = fullfile(f,d(i,:));
    dd=ls(ff);
    dd(1,:)=[];
    dd(1,:)=[];
    mkdir(fullfile(s,strcat('s',num2str(i))))
    mkdir(fullfile(t,strcat('s',num2str(i))))
    for j=1:ntrain
        b = strsplit(dd(j,:));
        I=imread(fullfile(f,a{1,1},b{1,1}));
        I=imresize(I,[227 227]);
        imwrite(I,fullfile(s,strcat('s',num2str(i)),
            strcat(num2str(j),'.jpg')));
    end

    for j=(ntrain+1):size(dd)
        b = strsplit(dd(j,:));
        I=imread(fullfile(f,a{1,1},b{1,1}));
        I=imresize(I,[227 227]);
        imwrite(I,fullfile(s,strcat('s',num2str(i)),
            strcat(num2str(j),'.jpg')));
        end
end
```

Detección de rostros para el problema 2, si no es capaz de detectar ninguno, esa imagen se descarta.

```
clc
clear all
close all
warning off
f='Carpeta_Origen';
d=ls(f);
d(1,:)=[];
d(1,:)=[];
g = 'Carpeta_Destino';
face_detector = vision.CascadeObjectDetector();
[n,m] = size(d);
for i=1:n
    a = strsplit(d(i,:));
    mkdir(fullfile(g,d(i,:)))
    ff = fullfile(f,d(i,:));
    dd=ls(ff);
    dd(1,:)=[];
    dd(1,:)=[];
    [nn,mm] = size(dd);
    k = 0;
    for j=1:nn
        b = strsplit(dd(j,:));
        I=imread(fullfile(f,a{1,1},b{1,1}));
        location_of_the_face = step(face_detector,I);
        if isvector(location_of_the_face)
            k = k+1;
            new_I = imcrop(I,location_of_the_face);
            new_I=imresize(new_I,[227 227]);
            imwrite(new_I,fullfile(g,a{1,1},strcat(
                num2str(k),'.jpg')));
        end
    end
end
```

III.II Basados en PCA

III.II.I PCA

Función para el cálculo del subespacio y las Eigenfaces.

```
function [T,m1,Eigenfaces,ProjectedImages,imageno] =
    Eigenface_calculation(imageno,nimages,f)
beforemean=[];
I=[];
T = imageno;
d=ls(f);
d(1,:)=[];
d(1,:)=[];
for i=1:imageno
    a = strsplit(d(i,:));
    ff = fullfile(f,d(i,:));
    dd=ls(ff);
    dd(1,:)=[];
    dd(1,:)=[];
    for j=1:nimages
        b = strsplit(dd(j,:));
        imagee = fullfile(f,a{1,1},b{1,1});
        imagg=imread(imagee);
        imagg = rgb2gray(imagg);
        imagg = imresize(imagg,[200,180],"bilinear");
        [m, n] = size(imagg);
        temp = reshape(imagg',m*n,1); % poner como
            vector
        beforemean = [beforemean,temp];
    end
    I = [I,mean(beforemean,2)];
    beforemean=[];
end
m1 = mean(I,2);
ima = reshape(m1',n,m);
ima = ima';
for i=1:T
    temp = double(I(:,i));
    I1(:,i)=(temp-m1);
end

a1 = [];
for i=1:T
```

```
37      te = double(I1(:,i));
38      a1 = [a1,te];
39  end
40  covv = a1'*a1;
41  [eigenvec,eigenvalue] = eig(covv);
42  d = eig(covv);
43  sorteigen = [];
44  eigval = [];
45  for i=1:size(eigenvec,2)
46      if d(i) > 0.5e+0.008; %prueba y error para ver con
            cuantos autovalores quedarse
47          sorteigen = [sorteigen,eigenvec(:,i)];
48          eigval = [eigval,eigenvalue(i,i)];
49      end
50  end
51  Eigenfaces = [];
52  Eigenfaces = a1*sorteigen;
53
54  for i=1:size(sorteigen,2)
55      k = sorteigen(:,i);
56      tem = sqrt(sum(k.^2));
57      sorteigen(:,i) = sorteigen(:,i)./tem;
58  end
59
60  Eigenfaces = a1*sorteigen;
61  ProjectedImages = zeros(T,T);
62  for i=1:T
63      ProjectedImages(:,i) = Eigenfaces'*I1(:,i);
64  end
65  end
```

Función de clasificación de nuevas imágenes.

```
function [accuracy] = Recognition(T,m1,Eigenfaces,
    ProjectedImages,imageno,f)
MeanInputImage = [];
ac = 0;
total = 0;
d=ls(f);
d(1,:)=[];
d(1,:)=[];
for i=1:size(d)
    a = strsplit(d(i,:));
    ff = fullfile(f,d(i,:));
    dd=ls(ff);
    dd(1,:)=[];
    dd(1,:)=[];
    for j=1:size(dd)
        total = total+1;
        b = strsplit(dd(j,:));
        fname = fullfile(f,a{1,1},b{1,1});
        InputImage = imread(fname);
        InputImage = rgb2gray(InputImage);
        InputImage = imresize(InputImage,[200,180],'
            bilinear');
        [m, n] = size(InputImage);
        Imagevector = reshape(InputImage',m*n,1);
        MeanInputImage = double(Imagevector)-m1;
        ProjectInputImage = Eigenfaces'*MeanInputImage;
        Euclideandistance = zeros(T,T);
        for n=1:T
            Euclideandistance(:,n) = ProjectedImages(:,n
                )-ProjectInputImage;
        end
        tem = [];
        for n=1:size(Euclideandistance,2)
            k = Euclideandistance(:,n);
            tem(n) = sqrt(sum(k.^2));
        end
        [MinEuclid,index] = min(tem);
        if index == i
            ac = ac+1;
        else
        i
        index
```

```
40            end
41        end
42    end
43    accuracy = ac/total
44    end
```

Programa principal

```
1  clear;
2  clc;
3  close all;
4  tic
5  imageno = 4; %numero de clases distintas
6  nimages = 20; %numero de imagenes por clase
7  f = 'Carpeta_Entrenamiento';
8  [T,m1,Eigenfaces,ProjectedImages,imageno] =
       Eigenface_calculation(imageno,nimages,f);
9  f = 'Carpeta_Test';
10 accuracy = Recognition(T,m1,Eigenfaces,ProjectedImages,
       imageno,f);
11 toc
```

III.II.II Fisherfaces

Clasificación por LDA

```
function [accuracy] = Recognition(T,m1,Eigenfaces,U,
    imageno,NF,f)
MeanInputImage = [];
ac = 0;
total = 0;
d=ls(f);
d(1,:)=[];
d(1,:)=[];
for i=1:size(d)
    a = strsplit(d(i,:));
    ff = fullfile(f,d(i,:));
    dd=ls(ff);
    dd(1,:)=[];
    dd(1,:)=[];
    for j=1:size(dd)
        total = total+1;
        b = strsplit(dd(j,:));
        fname = fullfile(f,a{1,1},b{1,1});
        InputImage = imread(fname);
        InputImage = rgb2gray(InputImage);
        InputImage = imresize(InputImage,[200,180],'
            bilinear');
        [m, n] = size(InputImage);
        Imagevector = reshape(InputImage',m*n,1);
        MeanInputImage = double(Imagevector)-m1;
        ProjectInputImage = NF'*MeanInputImage;
        Euclideandistance = zeros(T-1,T);
        for n=1:T
            Euclideandistance(:,n) = U(n,:)'-
                ProjectInputImage;
        end
        tem = [];
        for n=1:size(Euclideandistance,2)
            k = Euclideandistance(:,n);
            tem(n) = sqrt(sum(k.^2));
        end
        [MinEuclid,index] = min(tem);
        if index == i
            ac = ac+1;
        else
```

```
38              i
39              index
40              end
41          end
42  end
43  accuracy = ac/total
44  end
```

Programa principal

```
clear;
clc;
close all;
tic
imageno = 4;
nimages = 20;
f = 'Carpeta_Entrenamiento';
[T,m1,Eigenfaces,ProjectedImages,imageno,I,a1] =
    Eigenface_calculation(imageno,nimages,f);
ProjectedImages = ProjectedImages';
S_b = zeros(T,T);
S_w = zeros(T,T);
mu = mean(ProjectedImages,1);
for i = 1:imageno
    k = 1+(i-1)*nimages;
    aux = ProjectedImages(k:i*nimages,:);
    temp_S_b = mean(aux,1) - mu;
    S_b = S_b + nimages * (temp_S_b'*temp_S_b);
    S_w = S_w + nimages * cov(aux);
end
[V,~] = eig(S_b,S_w);
NF = Eigenfaces * V(:,2:end);
U = a1' * NF;
f = 'Carpeta_Test';
accuracy = Recognition(T,m1,Eigenfaces,U,imageno,NF,f);
toc
```

III.III Basados en Redes Neuronales

III.III.I Perceptrón Multicapa

Función tangencial sigmodial

```
function y = tg_sig(n)
m = length(n);
y = zeros(m,1);
for i=1:m
    if n(i)<-10
        y(i) = -1;
    elseif n(i)>10
        y(i) = 1;
    else
        y(i) = (exp(n(i))-exp(-n(i))) ./ (exp(n(i))+exp
            (-n(i)));
    end
end
end
```

Creación de imágenes como vector

```
clear ; close all ; clc ; warning off
tic
f='Carpeta_Entrenamiento';
d=ls(f);
d(1,:)=[];
d(1,:)=[];
[n,m] = size(d);
Imagenes = [];
for i=1:n
    clear IM
    IM = [];
    ds = ls(strcat(f,"\",d(i,:)));
    ds(1,:)=[];
    ds(1,:)=[];
    [x,y] = size(ds);
    for k=1:x
        clear INT_S
        I=imread(fullfile(f,strtrim(d(i,:)),strtrim(ds(k
            ,:))));
        %I = rgb2gray(I);
        [a,b] = size(I);
```

```
21          % Convertir la imagen a un vector fila a fila
22          INT_S = reshape(I',a*b,1);
23          IM = [IM;INT_S'];
24      end
25      Imagenes = [Imagenes;IM];
26  end
27  % Cada fila es una imagen, las x primeras son el primer
        sujeto
28  save Imagenes.mat
29  toc
```

Programa principal, con verificación tanto sobre datos de entrenamiento como de test, para comprobar que no se produce sobre entrenamiento.

```
clear ; close all; clc
tic
load Imagenes.mat;
nfaces = 40;
nimages = 9;
Imagenes_01 = double(Imagenes)/255;
PIM = Imagenes_01(1:(nfaces*nimages),:);


Q = size(PIM, 1);

T = [ones(1,nimages); -ones(nfaces-1,nimages)];
for i=1:(nfaces-1)
    T = [T [-ones(i,nimages); ones(1,nimages); -ones(
        nfaces-1-i,nimages)]];
end

rng('shuffle') % define semilla en funcion del instante
    actual

% Valores iniciales
n1 = 25; %Numero de neuronas de la capa oculta:
    variables
ep = 0.1;
% Pesos iniciales
W1 = ep*(2*rand(n1,size(PIM, 2)) - 1);
b1 = ep*(2*rand(n1,1) - 1);
W2 = ep*(2*rand(nfaces,n1) - 1);
b2 = ep*(2*rand(nfaces,1) - 1);

% Usar one vs all
alfa = 0.001;
for Epocas = 1:100
    for q = 1:Q
        q = randi(Q);
        % Propagacion de la entrada hacia la salida
        P = double(PIM(q,:)');
        a1 = tg_sig(W1*P + b1);
        a2 = tg_sig(W2*a1 + b2);
        % Retropropagacion de la sensibilidades
        e = T(:,q) - a2;
```

```
39            s2 = -2*diag(1-a2.^2)*e;
40            s1 = diag(1-a1.^2)*W2'*s2;
41            % Actualizacion de pesos sinapticos y
                  polarizaciones
42            W2 = W2 - alfa*s2*a1';
43            b2 = b2 - alfa*s2;
44            W1 = W1 - alfa*s1*P';
45            b1 = b1 - alfa*s1;
46            % Error Cuadratico
47            ec(q) = e'*e;
48        end
49        ecm(Epocas)= sum(ec)/Q;
50        if mod(Epocas,50) == 0
51            Epocas
52            plot(ecm)
53            pause(0.1)
54        end
55    end
56    plot(ecm)
57
58    %Verificacion sobre datos de entrenamiento
59    for q=1:Q
60        P = double(PIM(q,:)');
61        a1 = tg_sig(W1*P + b1);
62        [a iwin(q)] = max(W2*a1 + b2); % funcion de
              actIIIacion Max: cual de las neuronas finales
              esta mas actIIIa
63    end
64
65    y = [1*ones(1,nimages)];
66    for i=2:nfaces
67        y = [y [i*ones(1,nimages)]];
68    end
69    NumeroAciertos = sum(y==iwin)
70    PorcentajeAciertos = NumeroAciertos/(nfaces*nimages)
71
72    % Verificacion sobre datos de Test
73    total = 0;
74    rep = [];
75    f = 'Carpeta_Test';
76    d=ls(f);
77    d(1,:)=[];
78    d(1,:)=[];
79    for i=1:size(d)
80        a = strsplit(d(i,:));
```

```
81      ff = fullfile(f,d(i,:));
82      dd=ls(ff);
83      dd(1,:)=[];
84      dd(1,:)=[];
85      [m k] = size(dd);
86      rep = [rep;m];
87      for j=1:size(dd)
88          total = total+1;
89          b = strsplit(dd(j,:));
90          fname = fullfile(f,a{1,1},b{1,1});
91          InputImage = imread(fname);
92          %InputImage = rgb2gray(InputImage);
93          [m, n] = size(InputImage);
94          Imagevector = double(reshape(InputImage',m*n,1))
                ;
95          P = Imagevector/255;
96          a1 = tg_sig(W1*P + b1);
97          [x yobt(i,j)] = max(W2*a1 + b2);
98      end
99  end
100 yreal = [1:nfaces];
101 NumeroAciertos = 0;
102 for i=1:nfaces
103     NumeroAciertos = NumeroAciertos + sum(repmat(i,1,rep
        (i))==yobt(i,1:rep(i)));
104 end
105 accuracy = NumeroAciertos/total
106
107 toc
```

III.III.II Perceptrón+PCA

Función para la proyección de imágenes no utilizadas en el subespacio generado por Eigenfaces, pero que se usarán en el perceptrón.

```
function [ProjectInputImages] = Projection(T,m1,
    Eigenfaces,Imageno,nimages,f)
MeanInputImage = [];
ProjectInputImages = [];
d=ls(f);
d(1,:)=[];
d(1,:)=[];
for i=1:Imageno
    a = strsplit(d(i,:));
    ff = fullfile(f,d(i,:));
    dd=ls(ff);
    dd(1,:)=[];
    dd(1,:)=[];
    for j=(size(dd)-nimages+1):size(dd)
        b = strsplit(dd(j,:));
        fname = fullfile(f,a{1,1},b{1,1});
        InputImage = imread(fname);
        InputImage = rgb2gray(InputImage);
        InputImage = imresize(InputImage,[200,180],'
            bilinear');
        [m, n] = size(InputImage);
        Imagevector = reshape(InputImage',m*n,1);
        MeanInputImage = double(Imagevector)-m1;
        ProjectInputImage = Eigenfaces'*MeanInputImage;
        ProjectInputImages = [ProjectInputImages;
            ProjectInputImage'];
    end
end
end
```

Función que utiliza un perceptrón.

```
function [W1,W2,b1,b2] = perceptron(alfa,Q,PIM,T,Nepocas
    ,Nneur,nfaces)
rng('shuffle') % semilla aleatoria en funcion del
    instante actual

% Valores iniciales
```

```
5  ep = 0.1;
6  % Pesos iniciales
7  W1 = ep*(2*rand(Nneur,size(PIM, 2)) - 1);
8  b1 = ep*(2*rand(Nneur,1) - 1);
9  W2 = ep*(2*rand(nfaces,Nneur) - 1);
10 b2 = ep*(2*rand(nfaces,1) - 1);
11 for Epocas = 1:Nepocas
12     for q = 1:Q
13         % q = randi(Q);
14         % Propagacion de la entrada hacia la salida
15         P = double(PIM(q,:)');
16         a1 = tg_sig(W1*P + b1);
17         a2 = tg_sig(W2*a1 + b2);
18         % Retropropagacion de la sensibilidades
19         e = T(:,q) - a2;
20         s2 = -2*diag(1-a2.^2)*e;
21         s1 = diag(1-a1.^2)*W2'*s2;
22         % Actualizacion de pesos sinapticos y
               polarizaciones
23         W2 = W2 - alfa*s2*a1';
24         b2 = b2 - alfa*s2;
25         W1 = W1 - alfa*s1*P';
26         b1 = b1 - alfa*s1;
27     end
28 end
29 end
```

Función para la clasificación.

```
function [accuracy] = Recognition(nfaces,Eigenfaces,W1,
    W2,b1,b2,m1,f)
total = 0;
rep = [];
d=ls(f);
d(1,:)=[];
d(1,:)=[];
for i=1:size(d)
    a = strsplit(d(i,:));
    ff = fullfile(f,d(i,:));
    dd=ls(ff);
    dd(1,:)=[];
    dd(1,:)=[];
    [m k] = size(dd);
    rep = [rep;m];
    for j=1:size(dd)
        total = total+1;
        b = strsplit(dd(j,:));
        fname = fullfile(f,a{1,1},b{1,1});
        InputImage = imread(fname);
        InputImage = rgb2gray(InputImage);
        InputImage = imresize(InputImage,[200,180],'
            bilinear');
        [m, n] = size(InputImage);
        Imagevector = reshape(InputImage',m*n,1);
        MeanInputImage = double(Imagevector)-m1;
        ProjectInputImage = Eigenfaces'*MeanInputImage;
        P = ProjectInputImage/max(ProjectInputImage);
        a1 = tg_sig(W1*P + b1);
        [x yobt(i,j)] = max(W2*a1 + b2);
    end
end
yreal = [1:nfaces];
NumeroAciertos = 0;
for i=1:nfaces
    NumeroAciertos = NumeroAciertos + sum(repmat(i,1,rep
        (i))==yobt(i,1:rep(i)));
end
accuracy = NumeroAciertos/total;
end
```

Programa principal

```
clear;
clc;
close all;
%calcular eigenfaces: el numero de imagenes de datos
tic
imageno = 4;
nimages = 10;
f = 'Carpeta_Entrenamiento';
[T,m1,Eigenfaces,ProjectedImages,imageno,nimages] =
    Eigenface_calculation(imageno,nimages,f);

alfa = 0.01;
Nneur = 25;
nfaces = 4;
nimages = 20-nimages; % las no usadas en PCA
Nepocas = 1000;

Imagenes = Projection(T,m1,Eigenfaces,nfaces,nimages,f);
PIM = Imagenes/max(max(Imagenes));

Q = size(PIM, 1);

T = [ones(1,nimages); -ones(nfaces-1,nimages)];
for i=1:(nfaces-1)
     T = [T [-ones(i,nimages); ones(1,nimages); -ones(
         nfaces-1-i,nimages)]];
end

[W1,W2,b1,b2] = perceptron(alfa,Q,PIM,T,Nepocas,Nneur,
    nfaces);

f = 'Carpeta_Test';
accuracy = Recognition(nfaces,Eigenfaces,W1,W2,b1,b2,m1,
    f)
toc
```

III.III.III CNN

Uso de la red preentrenada para crear la red para estos rostros en particular

```
clc
clear all
close all
warning off

tic
g=alexnet;
layers=g.Layers;
layers(23)=fullyConnectedLayer(4); % el numero de imagenes distintas que se quieren clasificar
layers(25)=classificationLayer;
% en el primer string va sel nombre de la carpeta donde tienes las imagenes, pueden estar tanto en color como en blanco y negro, interpreta ambas
allImages=imageDatastore('Carpeta_Entrenamiento','IncludeSubfolders',true, 'LabelSource','foldernames');
allImages = augmentedImageDatastore([227,227],allImages,'ColorPreprocessing','gray2rgb');
% sgdm : stochastic gradient descent, el metodo que utiliza de entrenamiento
opts=trainingOptions('sgdm','InitialLearnRate',0.001,'MaxEpochs',20,'MiniBatchSize',64);
myNet1=trainNetwork(allImages,layers,opts);
save myNet1;
toc
```

Clasificación usando la red preentrenada

```
clc;close;clear
tic
load myNet1;
ac = 0;
k = 0;
f = 'Carpeta_Test';
d=ls(f);
d(1,:)=[];
d(1,:)=[];
for i=1:size(d)
    a = strsplit(d(i,:));
    ff = fullfile(f,d(i,:));
    dd=ls(ff);
    dd(1,:)=[];
    dd(1,:)=[];
    for j=1:size(dd)
        k = k+1;
        b = strsplit(dd(j,:));
        dir = fullfile(f,a{1,1},b{1,1});
        e=imread(dir);
        % e = cat(3, e, e, e); en caso de ser en blanco
            y negro
        label=classify(myNet1,e);
        aux = strsplit(string(label),'s');
        if str2num(aux(2))==i
            ac = ac+1;
        else
            i,j
        end
    end
end
ac/k
toc
```