Proyecto Fin de Máster
Master en Ingeniería Electrónica, Robótica y Automática

Integración y evaluación de sistemas de reidentificación de caras usando MTCNN y FaceNet en C++

Autor: Ana Belén Cordobés Menguiano
Tutor: Jesús Iván Maza Alcañiz

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Máster
Ingeniería de Sistemas y Automática

Integración y evaluación de sistemas de reidentificación de caras usando MTCNN y FaceNet en C++

Autor:
Ana Belén Cordobés Menguiano

Tutor:
Jesús Iván Maza Alcañiz
Profesor titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2019

Proyecto Fin de Máster: Integración y evaluación de sistemas de reidentificación de caras usando MTCNN y FaceNet en C++

Autor: Ana Belén Cordobés Menguiano

Tutor: Jesús Iván Maza Alcañiz

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2019

El Secretario del Tribunal

A mi familia

A mis maestros

A mis compañeros

Agradecimientos

En este apartado me gustaría agradecer a todas aquellas personas que me han ayudado, especialmente en este año. A todos mis profesores, por su capacidad de compartir, por su entusiasmo y su comprensión. En particular a Iván Maza, por su tiempo, su paciencia, su interés. A *4i*, por el tiempo dedicado, la confianza y la oportunidad que me han dado.

A mis padres y a mi hermano, que siempre me han apoyado en todo lo que he querido hacer. A mi familia, por que valoran lo que hago, aun si saber en muchas ocasiones de que se trata. A mis amigos y compañeros, porque me permiten abstraerme de casi cualquier cosa y disfrutar de casi cualquier momento.

Ana Belén Cordobés Menguiano

Sevilla, 2019

Resumen

En la actualidad el reconocimiento facial ha cobrado gran importancia debido a las múltiples aplicaciones con las que se le vincula. En este proyecto, junto con la empresa *4i*, se ha abordado el problema de conocer si un individuo ha estado en un centro comercial anteriormente en base a las imágenes de las cámaras de vigilancia. Para ello se han pretendido solucionar dos problemas mediante el uso de redes neuronales convolucionales. Por un lado, la detección de rostros en imágenes y, por otro, la identificación de estos. Para ello se han utilizado las redes neuronales *MTCNN* y *FaceNet*.

Además, se ha comparado la red utilizada para la detección de rostros (*MTCNN*) con una ofrecida por la librería *DLIB*, basada en *HOG* y *SVM*. Para ello se han utilizado dos bases de datos, *YouTube Faces Database* y *GRAZ-01*. Finalmente, se ha validado el funcionamiento de *FaceNet* con el primer dataset indicado.

Hay que indicar que, para todo ello, ha sido necesaria la implementación de detectores en el lenguaje de programación C++.

Abstract

At present, facial recognition has gained great importance due to the multiple applications with which it is linked. In this project, together with the company 4i, the problem of knowing if an individual has been in a shopping center previously based on the images of surveillance cameras has been addressed. To this end, two problems have been tried to be solved through the use of convolutional neural networks. On the one hand, the detection of faces in images and, on the other, the identification of these. For this, the neural networks MTCNN and FaceNet have been used.

In addition, the network used for face detection (MTCNN) has been compared with one offered by the DLIB library, based on HOG and SVM. For this, two databases, YouTube Faces Database and GRAZ-01, have been used. Finally, the operation of FaceNet has been validated with the first dataset indicated.

It should be noted that, for all this, the implementation of detectors in the C ++ programming language has been necessary.

-translation by google-

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

NN	Neural Network
CNN	Convolutional Neural Network
SVM	Support Vector Machines
HOG	Histogram of Oriented Gradients
SVM	Support Vector Machine
NMS	Non-Maximum Suppression
IoU	Intersection Over Union
P-Net	Proposal Network
R-Net	Refine Network
O-Net	Output Network

1 INTRODUCCIÓN

Una computadora puede ser llamada "inteligente" si logra engañar a una persona haciéndole creer que es un humano

Alan Turing

En los últimos años, el reconocimiento facial se ha convertido en un área de gran importancia debido a las múltiples aplicaciones que tiene. Entre ellas se encuentran aquellas relacionadas con el control de acceso a instalaciones o vehículos; el bloqueo o desbloqueo de elementos tecnológicos, como pueden ser *smartphones* o computadores; la prevención del fraude, especialmente en lo referido a la banca móvil; la prevención de ataques cibernéticos; o la video vigilancia avanzada. El hecho de que cada vez más sectores se interesen en ella contribuye a la existencia de gran cantidad de grupos de investigación buscando soluciones cada vez más exactas a dichos problemas.

Desde hace un tiempo, estos sistemas están llegándose a aplicar de forma masiva. Por ejemplo, en China se prevé que la cantidad de cámaras destinadas a ello llegue a las 400000. En la actualidad, en este país son usadas para la vigilancia de grupos étnicos minoritarios, la búsqueda de delincuentes y la monitorización de alumnos en los colegios. En 2017 la plataforma *Alipay*, de la empresa *Alibaba*, lanzó el servicio *Smile To Pay* [1] que permite el pago utilizando técnicas de reconociendo facial. En los Juegos Olímpicos de Tokio de 2020 se implantarán sistemas con un procesador *Intel Core i5* para la identificación de más de 300000 personas, entre ellas atletas y voluntarios. Todo esto está dando lugar a un gran número de detractores, que opinan que, aunque estos sistemas puedan tener gran cantidad de ventajas, provocan la pérdida de la privacidad del individuo. Además, se cree que también pueden ocasionar un aumento de delitos tales como el acoso. Dicho debate da lugar a que muchos gobiernos se planteen la prohibición o el establecimiento de estos sistemas. Un ejemplo de esto ocurrió en septiembre de 2019, cuando se aprobó la prohibición del uso de estas técnicas en las cámaras policiales.

1.1 Objetivo y motivación

EL objetivo de este proyecto es el de utilizar la red neuronal convolucional FaceNet para conseguir identificar individuos junto con el sistema de detección de rostros *MTCNN*. Para ello, se utilizará el lenguaje de programación C++ y la plataforma de aprendizaje automático TensorFlow. Se pretenderá estudiar el sistema propuesto para determinar su viabilidad cuando el número de individuos a considerar es elevado.

La motivación de su uso es la de considerar dicho sistema para extraer datos estadíscos acerca de la cantidad de veces que un individuo visita un determinado establecimiento.

1.2 Estado del arte

Como se ha indicado, el problema de identificación facial actualmente ha cobrado mucha importancia y se encuentra en continuo desarrollo. Este abarca multitud de campos, siendo la detección facial de vital importancia. La detección facial consiste en encontrar rostros dentro de una imagen en la que puede haber ninguna, una o varias caras. Uno de los algoritmos más relevantes, propuesto por Viola y Jones en [2], utiliza las características de Haar, obtenidas al recorrer una imagen realizando convoluciones con los filtros de la Ilustración 1, junto con la cascada de clasificadores AdaBoost (*Adaptative Boosting*). AdaBoost,presentado en 1995 en [3], es un clasificador compuesto por un conjunto de clasificadores débiles, en concreto, árboles de decisión de un solo nivel entrenados de forma iterativa. Uno de los principales problemas de la deteción automática es la clasificación de un amplio número de características, debido a su alto gasto computacional. Para ello, se propuso en [2] el uso de una cascada de clasificadores, que utiliza un pequeño número de características en sus primeras etapas, siendo aumentado progresivamente en las siguientes. El objetivo que tiene es el de descartar ventanas en las que no aparecen rostros con un número inferior de características. Las ventanas no descartadas pasan a la siguiente etapa, que con más características son clasificadas de la misma manera. Finalmente se obtiene un clasificador fuerte cuya generación se esquematiza en la Ilustración 2.

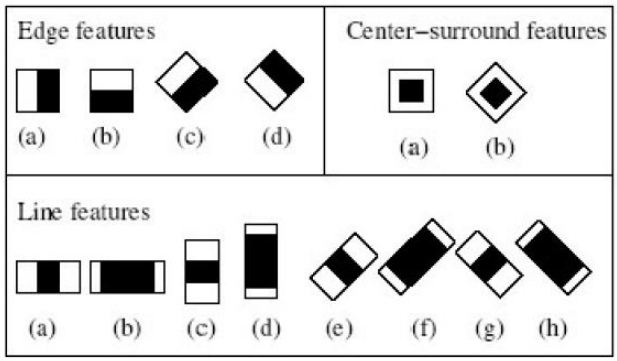

Ilustración 1. Conjunto extendido de filtros de Haar.

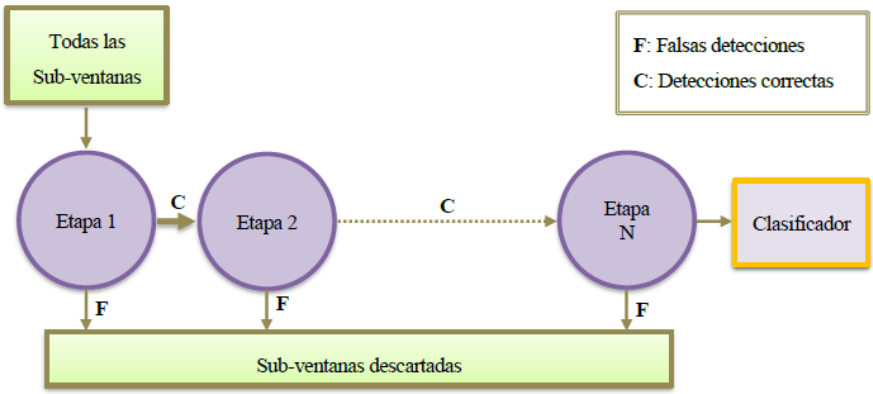

Ilustración 2. Cascada de clasificadores.

Este método tubo gran relevancia, aunque su funcionamiento no es adecuado con rostros que no se presentan frontalmente en la imagen y, por tanto, también genera problemas ante aspectos tales como oclusiones.

$$Valor\ del\ pixel\ central \begin{cases} 1 & si\ el\ pixel\ vecino \geq pixel\ central \\ 0 & si\ el\ pixel\ vecino < pixel\ central \end{cases}$$

Ecuación 1. Cálculo del descriptor LBP.

Más tarde, en [4] se plantea el uso de patrones locales binarios (*LBP*) junto con el clasificador del vecino más cercano. *LBP* es un descriptor que, como se muestra en la Ilustración 3, se obtiene al realizar operaciones sobre los pixeles relativos a una ventana de vecindad, definida según la Ecuación 1. Con la matriz obtenida se obtiene un número binario, que da lugar a otro decimal, y que se corresponde con la característica obtenida. Esta tendrá información no solo sobre el píxel sobre el cual se realiza el cálculo, sino que también sobre sus vecinos. Todo el proceso se debe realizar sobre todos los pixeles de la imagen, es decir, se mueve la ventana por la fotografía mientras que se calcula el valor que la caracteriza. Estos valores dan lugar a una imagen LBP, cuyo histograma normalizado ocasionan el descriptor utilizado. Sobre este proceso se realizan posteriormente una serie de modificaciones que harán al descriptor más robusto ante aspectos relativos al ruido.

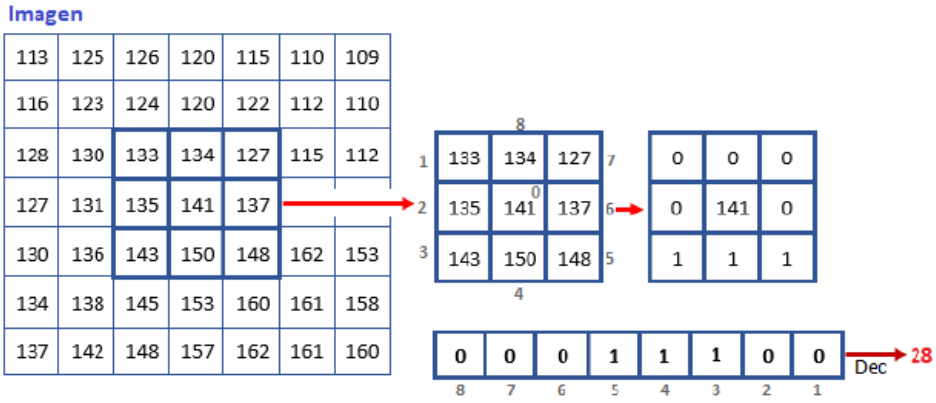

Ilustración 3. Ejemplo del cálculo del descriptor LBP.

En [5] se propone el uso de *SVM* para la clasificación en un sistema de detección facial mediante el reconocimiento de ojos en la imagen. En [6] se aplica el descriptor *HOG* y *LBP* junto con el clasificador *SVM* para la detección humana cuando hay oclusiones. En la sección 4.1 se explica brevemente el funcionamiento de *LBP* y *HOG*.

En la actualidad se utilizan habitualmente redes neuronales convolucionales para la detección de rostros. Una de estas es *MTCNN* [7], tratada en el apartado 4.2.

Técnicas de reconocimiento facial empiezan a desarrollarse a finales del siglo XX, pero no es hasta comienzos del siglo XXI cuando se comienzan a obtener buenos resultados. En 1991, en [8] se desarrolla un sistema de reconocimiento facial utilizando *EigenFaces* en el que se trata dicho problema matemáticamente utilizando PCA (*Principal Component Analysis*). Gracias a esto se consigue representar un rostro como un vector de características de bajas dimensiones. Sin embargo, se generaban problemas en el reconocimiento ante cambios en la imagen, o en el rostro. Más tarde, en 1997, en [9], se desarrolla *FisherFaces* que utiliza el Discriminante Lineal de Fisher y *PCA*. Este tiene en cuenta la reflexión de la luz, funcionando con diferentes condiciones de iluminación. Posteriormente, se comienzan a utilizar descriptores locales como HOG, utilizado en [10] para la detección de personas, o LBP, descrito en [11] y [12].

En [13] se utiliza un método denominado *Representación dispersa* o, en inglés, *Sparse Representation*, que tiene como objetivo combatir los problemas ocasionados por los efectos de oclusión, iluminación y ruido. Para obtener un buen resultado es crítico que el número de características sea lo suficientemente elevado, mientras que la elección de estas deja de tener tanta importancia. Su funcionamiento consiste en considerar una imagen, ya sea con oclusiones o ruido, como la suma de la combinación lineal entre el conjunto de las imágenes de entrenamiento y unos coeficientes más las oclusiones o ruido aleatorio.

En [14] se utiliza un método llamado *Aprendizaje métrico* o, en inglés, *Metric Learning*. Este pretende optimizar una matriz de manera que para un mismo individuo la distancia entre sus características de entrenamiento y validación sea mínima y, para distintos, sea máxima. Dicha distancia viene determinada por la Ecuación 2. Para conseguir un correcto funcionamiento el tamaño del conjunto de muestras de entrenamiento debe ser elevado y se deben escoger bien los descriptores usados y la dimensionalidad de la matriz M.

$$d(x_e, x_v) = (x_e - x_v)^T M (x_e - x_v)$$

Ecuación 2. Cálculo de la distancia entre las características de entrenamiento y validación en la técnica de aprendizaje métrico.

Actualmente, se utilizan redes neuronales convolucionales para solucionar este problema. Una de estas es FaceNet [15], vista en el apartado 5.1.

En el Face Recognition Grand Challenge (*FRGC*) [16] se realiza un estudio acerca de los avances en el rendimiento de los sistemas de reconocimiento de los últimos años. Como resultados, se obtiene que las mejoras con respecto a los años anteriores a 2002 son un orden de magnitud más precisos, y dos órdenes con respecto a los de 1995.

1.3 Estructura del Proyecto

Para realizar este proyecto se ha partido del directorio de trabajo ofrecido por la empresa *4i*. Se considera importante describir su estructura para posteriormente comentar que se ha implementado.

En la Ilustración 4 se puede ver la distribución que sigue. La carpeta *librerías* contiene aquellas librerías que son necesarias para la ejecución de muchos programas. Por ejemplo, dentro de ella se encuentran *OpenCV*, *Dlib*, *TensorFlow* y *OpenBlas*. En *common* hay ficheros comunes para las implementaciones de los detectores que hay programados. En *configurations* se encuentra la configuración de distintas cámaras, por lo que no será necesario su uso para la funcionalidad que se va a desarrollar. En *data* hay estructuras de datos que son utilizados para las salidas de los detectores. En *detector* se encuentran los detectores, además de las carpetas *test* y *testFaceNet*, que contienen un fichero principal con el cual pueden ser probados los detectores implementados. En *logger* se añaden funciones con el objetivo de registrar el estado por el que pasa el sistema y almacenarlo en documentos. En *stats*, se introducen distintas funcionalidades relativas, por ejemplo, a contadores y

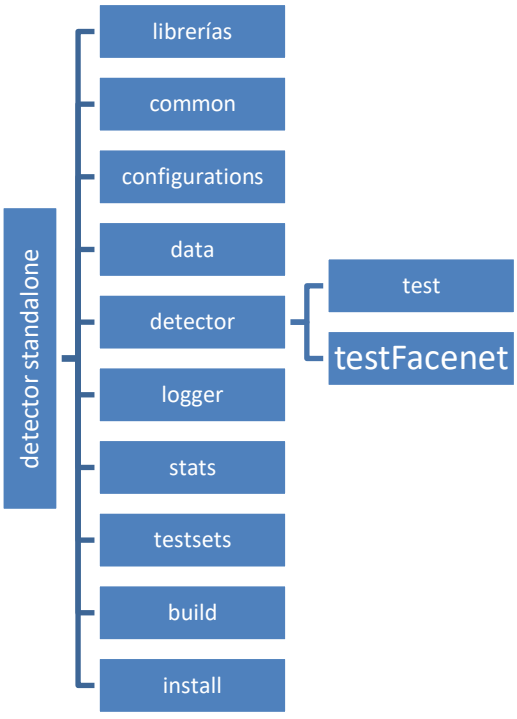

Ilustración 4. Estructura del proyecto.

temporizadores. En *test sets* hay conjuntos de imágenes para utilizar de prueba. La carpeta *build* se ha utilizado para almacenar los binarios obtenidos tras usar *cmake* y la carpeta *install* para instalar todos los archivos. Dentro de *detector_standalone*, al igual que de los subdirectorios, se encuentra un archivo *CMakeList.txt*, con reglas sobre la compilación.

A la carpeta *detector* han sido añadidos un total de 8 archivos. Estos son:

- pbox.cpp
- pbox.h
- network.cpp
- network.h
- detector_mtcnn.cpp
- detector_mtcnn.h
- detector_mtcnn_facenet.cpp
- detector_mtcnn_facenet.h

Los ficheros indicados están basados en [17]. Los 4 primeros: *pbox.cpp*, *pbox.h*, *network.cpp* y *network.h* no han sido modificados y se encargan, respectivamente, de la implementación y definición de estructuras necesarias para facilitar la creación de la red neuronal MTCNN y de la implementación de la misma. Los archivos *detector_mtnn.cpp* y *detector_mtcnn.h* contienen la programación de la red neuronal *MTCNN* intentando mantener la forma del resto de descriptores dados por *4i*. Los archivos *detector_mtcnn_facenet.cpp* y *detector_mtcnn_facenet.h* contienen además el cálculo del vector de características asociado a cada uno de los rostros detectados.

Además, se ha creado un subdirectorio, llamado *testFacenet*, que contiene un *main* que nos permite probar la clasificación de rostros, dada una imagen, en función de las características previamente obtenidas y almacenadas en un *JSON*.

Más adelante, se explicará más detalladamente el comportamiento programado para cada uno de estos ficheros.

Con respecto a la estructura que se va a seguir en este documento, se ha visto necesario comentar, en el capítulo 2, el sistema operativo (sección 2.1) y el software utilizado, compuesto por los programas *CMake* (sección 2.2), *TensorFlow* (sección 2.3), *OpenBlas* (sección 2.4) y *Git* (sección 2.5); seguidamente, se explica superficialmente el funcionamiento de las redes neuronales (capítulo 3), centrándose en aspectos básicos de estas en la sección 3.1 y, en la sección 3.2, en las redes neuronales convoluciones. El siguiente tema que se tratará se centra en la detección de rostros (capítulo 4), y por tanto involucra a uno de los métodos propuestos por *DLIB* (sección 4.1), y dados por *4i*, a la *CNN MTCNN* (sección 4.2) y a una comparación entre ambas (sección 4.3). El siguiente punto, se centra en el reconocimiento facial (capítulo 5), donde se explica el funcionamiento de la red neuronal *FaceNet* (sección 5.1) y se realiza una prueba de su funcionamiento (sección 5.2). En el capítulo 6 se detallan los detectores que han sido implementados, comenzando en la sección 6.1 por *MTCNN*, seguida en la sección 6.2 por su unión con FaceNet y terminando en la sección 6.3 con el fichero principal encargado de administrar la llamada a este último clasificador. Finalmente, en el capítulo 7, se comentan las conclusiones sacadas junto con el trabajo futuro asignado al proyecto.

2 SOFTWARE EMPLEADO

El hardware es lo que hace a una máquina rápida; el software es lo que hace que una máquina rápida se vuelva lenta.

Craig Bruce

En este proyecto se va a utilizar el sistema operativo Ubuntu en su versión 18.04 LTS y el lenguaje de programación C++. Para su compilación se ha utilizado la herramienta CMake, junto con los comandos *make* y *make* install. Para la implementación de la red neuronal *MTCNN* se ha obtenido el código de [17]. Con el objetivo de optimizar su ejecución se va a utilizar *OpenBlas*. Para la red neuronal *FaceNet* se va a usar la plataforma *TensorFlow* con la que se montará la red, que está previamente entrenada y cuyos parámetros se han obtenido de [18]. Además, con el objetivo de trabajar conjuntamente sobre el espacio de trabajo de la empresa *4i* se ha utilizado el programa de control de versiones *GIT*.

2.1 Ubuntu

Como se ha indicado anteriormente, para realizar este Proyecto se ha utilizado el sistema operativo (S.O) Ubuntu en su versión estable 18.04. Ubuntu es un sistema operativo de código abierto que tiene como núcleo (*kernel*) la distribución Linux, a su vez basada en *Debian*. En la Ilustración 5 se muestran los logos de Ubuntu y Debian.

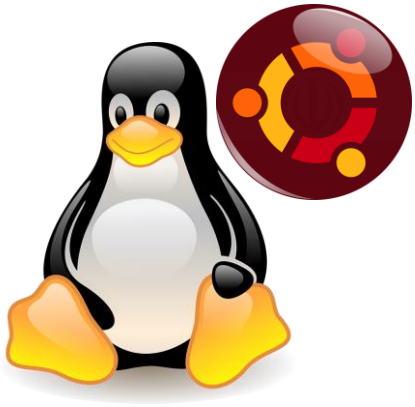

Ilustración 5. Logos de Debian (izquierda) y ubuntu (derecha).

Su sistema de archivos se encuentra organizado en una estructura de árbol según se observa en la Ilustración 6. Es interesante comentarla debido a que se necesita acceder a ficheros de configuración de las redes neuronales de forma adecuada, con el objetivo de que sea independiente del computador en el que sea ejecutado el programa.

Sin entrar en mucho detalle, se parte de un directorio raíz ("/"), bajo el que cuelgan distintas secciones, tales como */bin*, con comandos básicos de usuario, como *chmod, kill, grep, ls*…; */etc*, con archivos de configuración, como *hosts, profile*...; */sbin*, con binarios esenciales del sistema, como *reboot, ifconfig*…; */usr*, con datos de soporte, solo de lectura, para aplicaciones de usuario; */home*, donde se encuentran los directorios personales del usuario, y sobre la que cuelga la carpeta del proyecto; */lib*, con librerías; y */opt*, con aplicaciones adicionales instaladas.

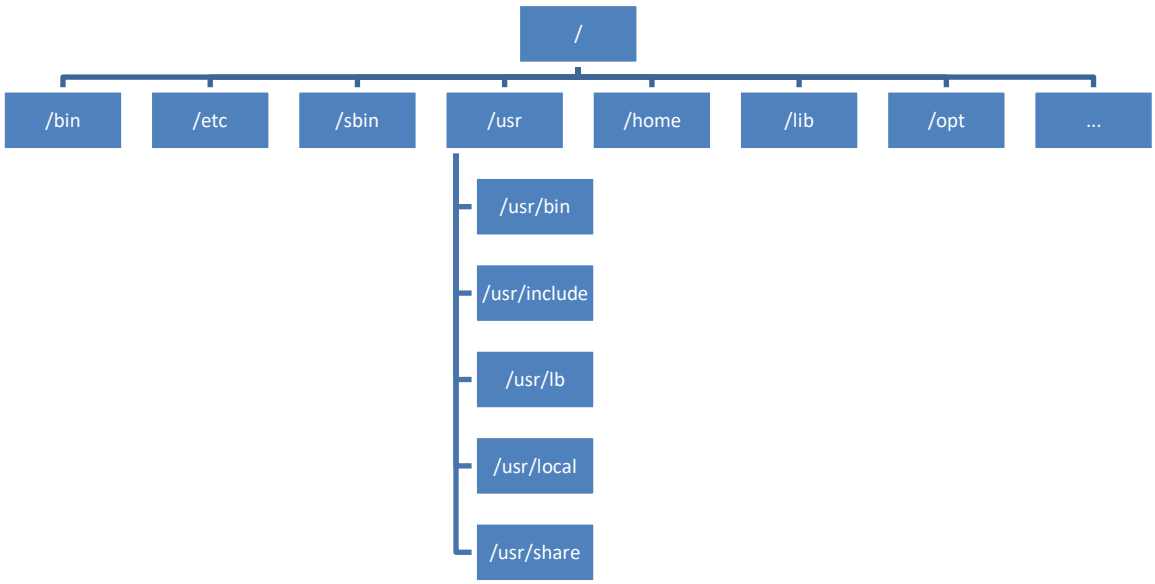

Ilustración 6. Sistema de archivos del sistema operativo Ubuntu

La ruta de un fichero es una dirección única a este mismo dentro del sistema de archivos de dicho *S.O*. Una ruta puede ser absoluta o relativa. Una ruta absoluta es aquella que parte del directorio raíz, y por tanto siempre comienza con "/". Una ruta relativa parte de una determinada carpeta y, por tanto, nunca debe comenzar con "/". Para referirnos a ficheros dentro del proyecto se van a utilizar rutas relativas, que serán pasadas a absolutas gracias a las librerías de *C++ "std::filesystem::Path"* y "*boost::filesystem::canonical*" que nos permiten, entre otras cosas, convertir una cadena en una ruta para saber si esta es absoluta y, de no serlo, la modifican para que lo sea, según el sistema de archivos del computador sobre el cual se está ejecutando.

2.2 CMake

Con el objetivo de compilar este Proyecto se van a utilizar las herramientas *make,* que compila el código fuente, a partir de archivos *MakeFiles*, creando los ejecutables asociados a la aplicación, y *make install*, que mueve los ficheros necesarios a un directorio adecuado. Para poder utilizar estos comandos es necesario generar los MakeFiles, motivo por el cual se utilizará la herramienta *CMake* [19].

CMake es una herramienta multiplataforma de código abierto diseñada para construir software, principalmente en C++. Su función, como se ha comentado, es crear archivos *MakeFiles* nativos automáticamente, que permitan controlar el proceso de compilación, a partir de otros ficheros de configuración. Estos últimos reciben el nombre de *CMakeList.txt* y se encuentran en cada uno de los directorios y subdirectorios asociados al proyecto. Además, contienen información no solo de los ficheros, sino que tambien de las librerías externas necesarias para su ejecución.

En la Ilustración 7 se puede ver la interfaz de *CMake*. Para este proyecto, como se observa, se ha incluido la construcción de los archivos en una carpeta llamada *build* y su instalación en otra denominada *install*. Las librerías externas de encuentran en otra carpeta *argo_libs_skylane_gpu*, por lo que, según se especifica en los *CMakeList.txt*, cuando se desee buscar alguna librería, como Tensorflow_CC, se hará automáticamente sobre el directorio de dicha carpeta. Una vez se han generado los *MakeFiles* con esta aplicación, dentro de la carpeta *build* del proyecto de deberán escribir los siguientes comandos:

make

make install

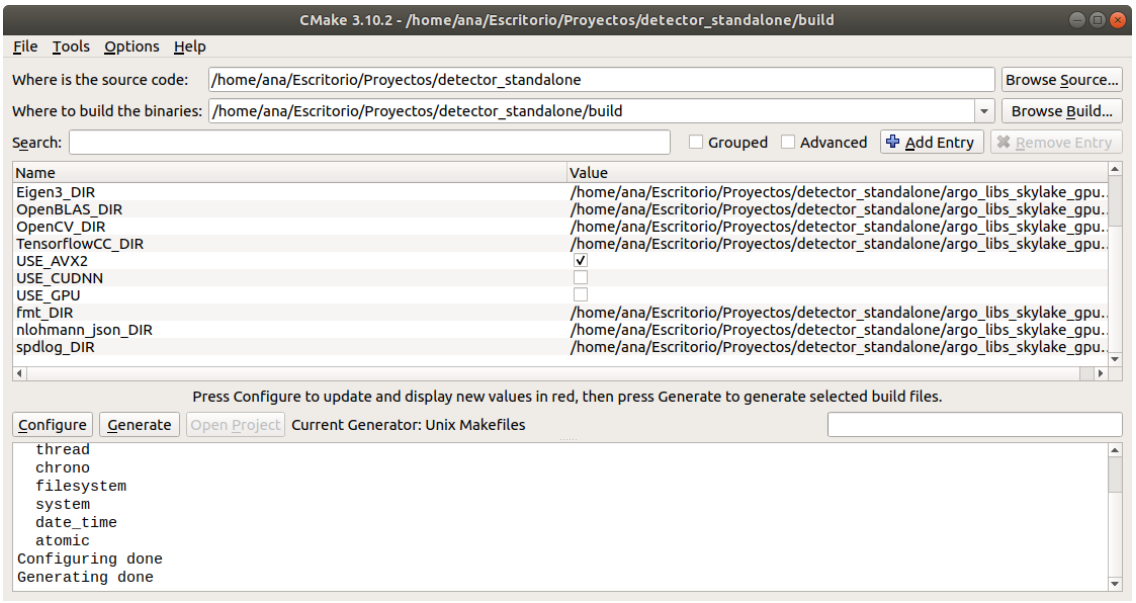

Ilustración 7. CMake-GUI

2.3 TensorFlow

Una versión anterior a *TensorFlow*, *DistBelief*, fue creada en 2011 por *Google Brain*, para realizar tareas de aprendizaje automático basadas en redes neuronales. Su uso estuvo presente en múltiples sectores, ya sea para aplicaciones comerciales o de investigación.

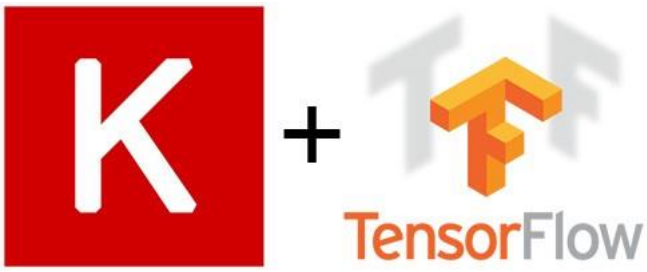

Ilustración 8. Logo de Keras y TensorFlow.

El 9 de noviembre de 2015 es lanzado *TensorFlow* una biblioteca de código abierto para computación numérica y, la segunda versión de *DistBelief*. Opera sobre diferentes plataformas como CPUs, GPUs, TPUs e incluso la placa de ordenador reducida (*SBC*) Raspberry Pi. Funciona con los sistemas operativos Linux de 64 bits, macOS y otras plataformas móviles con Android y iOS. Generalmente es usada con el lenguaje de programación *Python*, aunque también proporciona *APIs* para *C++*, *Java*, *Haskell*, *Go* y *Rust* entre otras. *TensorFlow* permite la construcción y el entrenamiento de redes neuronales, para lo cual realiza operaciones sobre elementos denominados tensores, que son *arrays* multidimensionales de datos.

Keras es una biblioteca de redes neuronales de código abierto programada en *Python*, que permite la construcción de redes neuronales convolucionales y recurrentes. En 2017 se integra en *TensorFlow*, lo que permite su uso con esta última. Este hecho dio lugar a que *TensorFlow* pasase a ser la biblioteca más utilizada en este campo. En la Ilustración 8 se aprecia el logo de *Keras* (a la izquierda) y de *TensorFlow* (a la derecha).

2.4 OpenBLAS

OpenBLAS es una biblioteca de código abierto que implementa la API BLAS y que contiene programas de algebra lineal básica optimizados para tipos de procesadores específicos. Su versión inicial se realiza en marzo de 2011 pero no se produce un lanzamiento estable hasta abril de 2019. Funciona en los sistemas operativos *Linux*, *Microsoft Windows*, *macOS* y *FreeBSD*. Se utiliza, en este proyecto, para la realización de las convoluciones ocasionadas en la red neuronal *MTCNN*.

2.5 Git

Git [20] es un Sistema de control de versiones distribuido de código abierto pensado para trabajar conjuntamente en proyectos de diversas dimensiones.

Ilustración 9. Logo de GIT [40].

En Git se tienen dos repositorios, uno local, y que abarca a los archivos que físicamente se almacenan en el computador, y otro remoto. Generalmente, en el inicio de cualquier proyecto, se comienza copiando archivos desde el repositorio remoto hasta el local. Para ello se utiliza la siguiente instrucción:

git clone <url del repositorio>

Para añadir nuevas funcionalidades al proyecto se suelen crear ramas, que nos permiten tener nuevas líneas de desarrollo independientes. Es similar a tener un espacio de trabajo idéntico, sobre el cual realizar cambios. En Git se recomienda tener 3 tipos de ramas. Estas son las que se visualizan en la Ilustración 10. Por un lado, está la rama principal, conocida como *master*, que debe ser estable. Consiguientemente está la rama *development*, donde se añaden todas las nuevas funcionalidades. Una vez que esta rama es probada y corregida se agrupa en *master*. Finalmente están las ramas *features* que contienen cada una de las nuevas funcionalidades de *development* por separado.

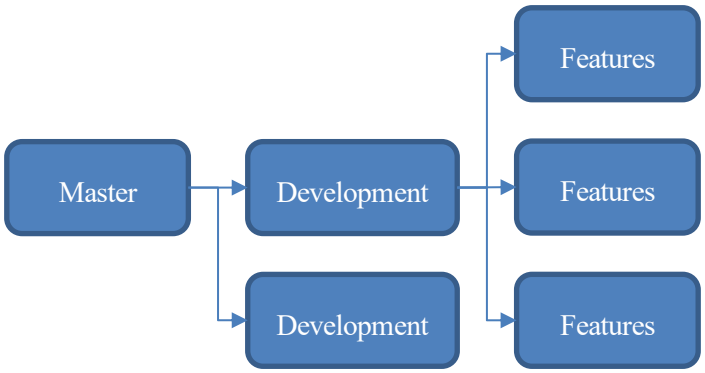

Ilustración 10. Tipos de ramas

Para crear una nueva rama y saltar a ella se utiliza el comando que viene a continuación:

git checkout -b <nombre de la nueva rama>

Además, se pueden conocer las ramas locales existentes con el comando:

git branch

Cuando se trabaja en el ordenador, los cambios realizados se vinculan al repositorio local. Estos no se verán reflejados en el remoto hasta que no sean subidos. Con el comando

git status

se pueden conocer los cambios efectuados localmente con respecto al repositorio remoto. De esta manera, pueden ser añadidos o ignorados. Además, se permite la eliminación de archivos. Los comandos usados para ello son los siguientes:

git add <nombre del archivo>

git checkout <nombre del archivo>

git rm <nombre del archivo>

Una vez han sido tratados los cambios estos deben ser confirmados, para ello se indica lo siguiente:

git commit -am <mensaje explicativo>

Finalmente, para subir los cambios al servidor remoto se realiza lo siguiente:

git push origin <nombre de la rama>

Como ya se ha indicado anteriormente, cuando una rama está correctamente comprobada, suele ser impactada con la anterior. Para ello, se utiliza el siguiente comando:

git merge <nombre de la rama>

3 INTRODUCCIÓN A LAS REDES NEURONALES

Las 'leyes del pensamiento' no solo dependen de las propiedades de las células cerebrales, sino del modo en que están conectadas.

La sociedad de la mente (1987)

En este capítulo se va a dar una breve introducción a los conceptos de redes neuronales necesarios para comprender los pilares en los que se basa el trabajo realizado. Se comenzará con una descripción de las redes neuronales, así como del ámbito actual en el que se encuentran (*Deep Learning*) para proseguir con los conceptos pertinentes a su entrenamiento. Posteriormente, se hablará de redes neuronales convolucionales, comúnmente usadas para la detección en imágenes de personas, animales u objetos.

3.1 Redes neuronales (*NN – Neural Networks*)

Las redes neuronales artificiales son sistemas computacionales principalmente englobados en el campo del aprendizaje profundo, o en inglés, *Deep Learning*. Están compuestas por un conjunto de unidades, llamadas neuronas, que realizan cálculos simples, con el objetivo de, pasadas unas ciertas capas de estas, dar una respuesta a un problema específico. En la actualidad han cobrado una gran importancia, llegándose a solucionar problemas relacionados con el reconocimiento de imágenes, de caracteres, control de vehículos y generación de trayectorias, prevención de fraudes y análisis genético, entre otros.

3.1.1 Inteligencia artificial, Machine Learning y Deep Learning.

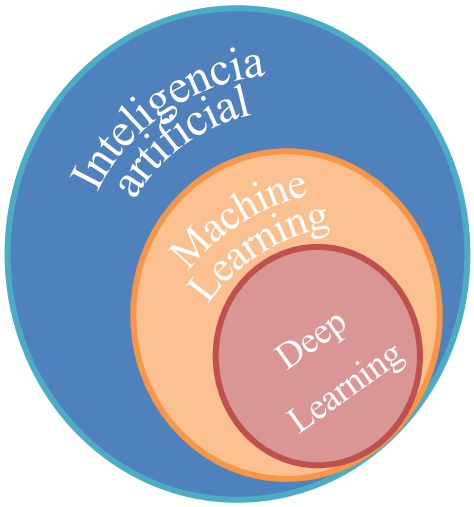

Ilustración 11. Inteligencia artificial, machine learning y deep learning.

El *Deep Learning* o aprendizaje profundo es una metodología de aprendizaje automático (*Machine Learning*) perteneciente al campo de la inteligencia artificial. El objetivo de la inteligencia artificial es el de dotar a las máquinas de la capacidad de comportarse de manera inteligente utilizando diversos mecanismos y programas. Este término comienza a utilizarse en la década de los 50s, sin embargo, termina siendo un área que no aporta buenos resultados debido a la inexistencia de elementos con alta capacidad computacional y de conjuntos elevados de datos. Es en la década de 2010 cuando se considera que explota, debido al desarrollo de unidades de procesamiento gráfico (*GPUs*), capaces de aportar la potencia de cálculo necesaria y, a la importancia que ha cobrado el *BigData*. En el 2012, la red neuronal convolucional *AlexNet* [21], entrenada con una *GPU*, gana el reto *ImageNet Large Scale Visual Recognition Challenge* con una *CNN* compuesta por 5 capas convolucionales, de las cuales algunas están seguidas de una etapa de submuestreo, y tres capas *fully connected* (Ilustración 12). La función de activación utilizada fue la ReLU, por dar lugar a un entrenamiento más rápido que otras, como la *tanh*. Era capaz de detectar un total de 1000 clases, entre las que se encuentran gatos, personas, cerezas…

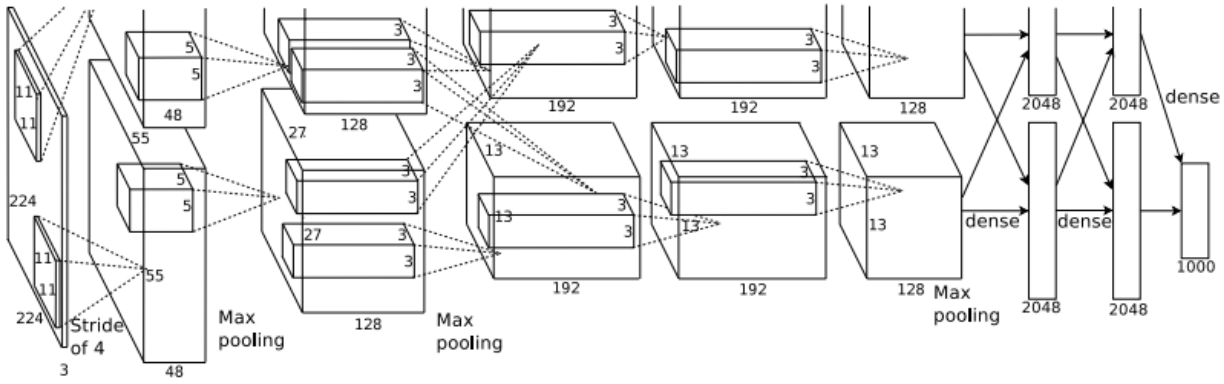

Ilustración 12. Arquitectura de la red neuronal convolucional AlexNet. Imagen tomada de [21].

Esto supuso un antes y un después en lo relativo a las *CNN*, dando lugar a que se utilizaran en otros campos, como los referidos al reconocimiento de rostros. La red neuronal FaceNet [15] permite la obtención de 128 características por rostro, gracias a lo cual, utilizando la distancia euclidea, nos permite conocer si el individuo de una imagen se corresponde con el individuo de otra. En el apartado 5.1 se explicará detalladamente su funcionamiento.

El *machine learning* consiste en aportar esta inteligencia utilizando algoritmos que permitan a la máquina aprender a partir de un conjunto de datos agrupados en patrones, sin necesidad de que se haya realizado una programación explícita. El Deep Learning, trata de realizar todo esto, imitando el cerebro humano, es decir, la red de neuronas que tiene. Este último, por tanto, sería lo más parecido a la inteligencia humana.

3.1.2 Inspiración biológica

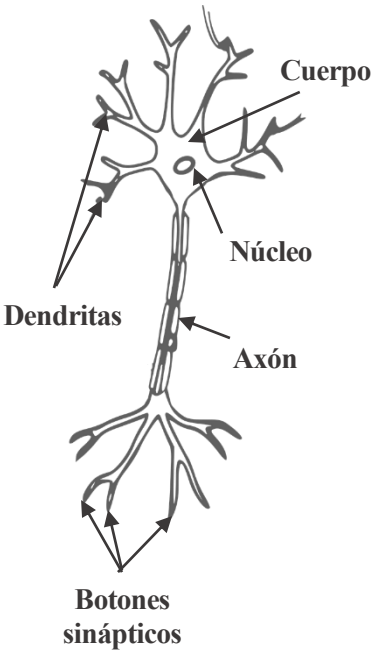

Ilustración 13. Representación de una neurona biológica.

Las redes neuronales artificiales se inspiran en aquellas presentes en el cerebro humano, por ser este un elemento complejo, no lineal y paralelo de procesamiento de información. Estas están compuestas por una agrupación de neuronas que establecen conexiones sinápticas consistentes en un intercambio eléctrico entre el botón sináptico de una neurona y el dendrítico de otra. Cuando se produce una estimulación en una neurona esta se altera electroquímicamente, almacenando energía que, al alcanzar un determinado valor, se descarga a través del Axón, partiendo del cuerpo hasta llegar a sus neuronas vecinas, creando así una conexión. El cerebro tiene aproximadamente 100G neuronas, cada una de las cuales establece 10k conexiones. Parte de la estructura de estas está determinada en el nacimiento, mientras que otra parte se desarrolla durante el aprendizaje, momento en el que se crean nuevas conexiones y se debilitan o pierden otras.

3.1.3 El perceptrón

Basándose en la idea anterior, Frank Rosenblatt presenta el 1958 el perceptrón [22], que puede ser considerado como una neurona o como la estructura de red neuronal más simple, cuyo modelo puede verse en la Ilustración 14. Este posee un número determinado de entradas (X) que son análogas a las dendritas de las neuronas biológicas; para las cuales hay asociados diferentes pesos (W), similares a la intensidad con la que se produce la sinapsis; una unión sumadora; una función de activación (g), parecida a la realizada en el cuerpo o soma; y una salida, que modelaría el axón.

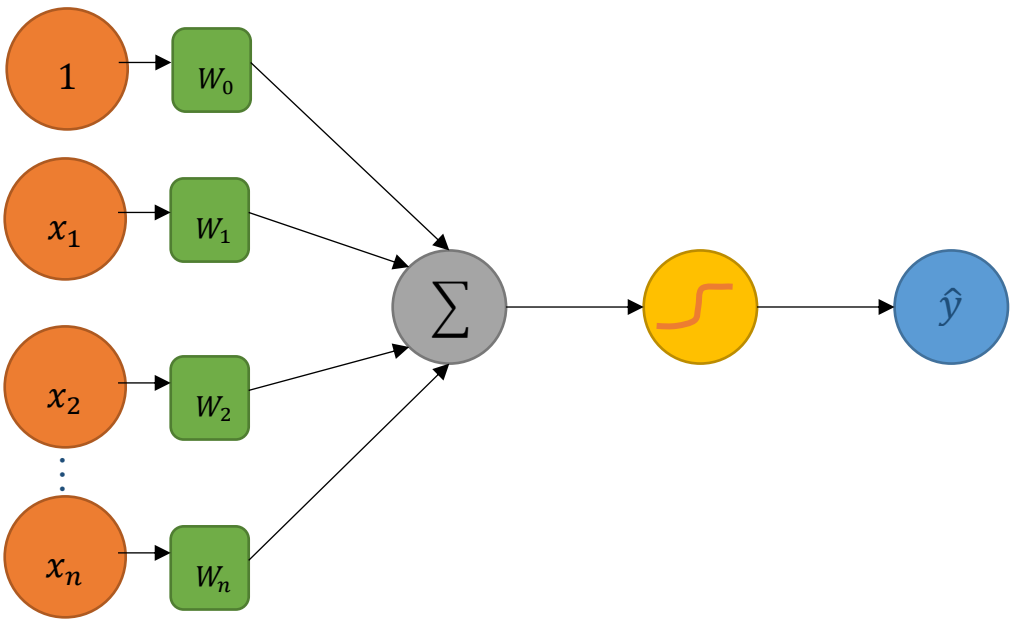

Ilustración 14. Modelo de perceptrón.

La expresión que modela la salida del perceptrón se aprecia en la Ecuación 3.

$$\hat{y} = g(w_0 + \sum_{i=1}^{n} x_i w_i)$$

Ecuación 3. Salida de un perceptrón.

La función de activación (g) puede ser de muchos tipos, siendo las más comunes la función lineal (ReLU), la función sigmoide y la tangente hiperbólica (Ilustración 16). La función sigmoide mantiene la salida en el intervalo 0 y 1, y cobra especial importancia cuando se quieren obtener salidas relacionas con distribuciones de probabilidad. La función ReLU tiene como ventaja el hecho de que es sencilla de computar ya que a partir de 0 es una función lineal igual a la entrada, siendo nula en todo momento anterior. La finalidad de una función de activación es la de introducir no linealidades a la red. La importancia de esto estriba en que generalmente la separabilidad de dos clases no puede realizarse mediante una línea recta, pero si mediante una curva. Un ejemplo de esto puede verse en la Ilustración 15, donde la clase *círculos* y la clase *triángulos* no son separables mediante rectas.

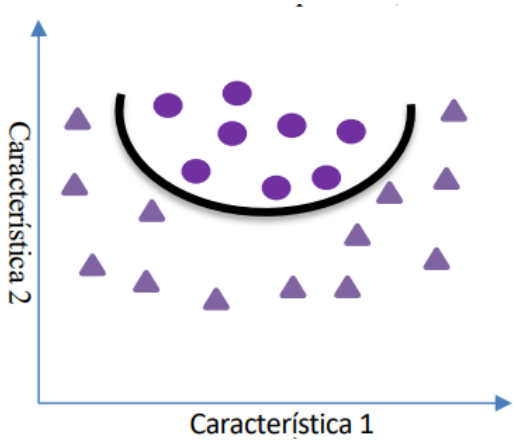

Ilustración 15. Clases no linealmente separables.

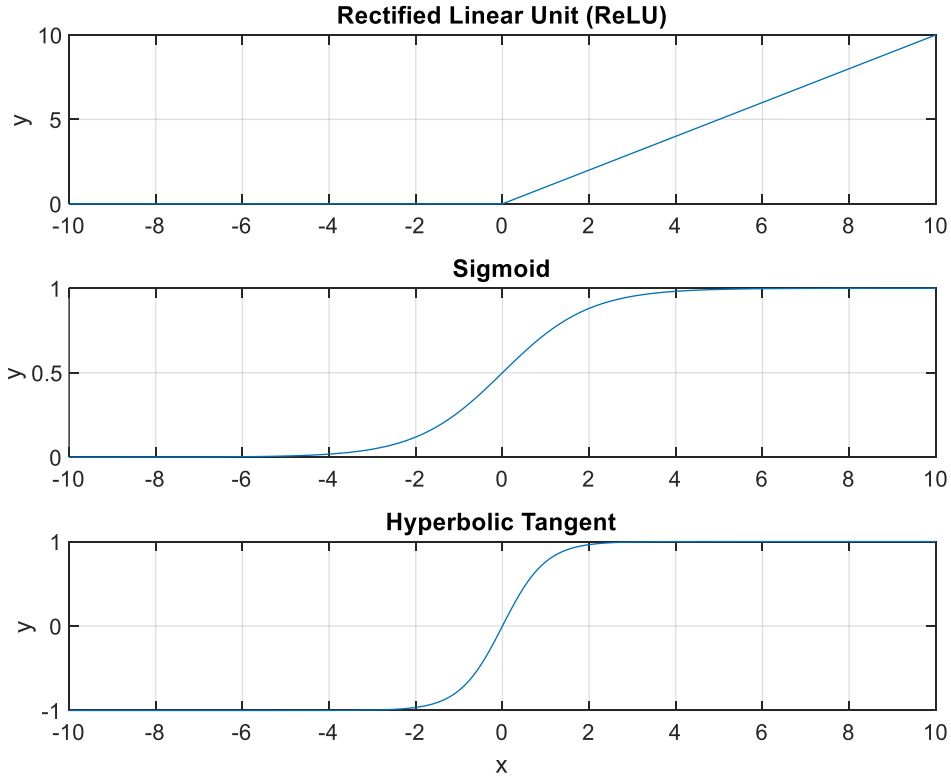

Ilustración 16. Ejemplo de funciones de activación.

3.1.4 Redes de neuronas como agrupación de perceptrones

Con una neurona como la que se ha visto pueden no obtenerse buenos resultados para muchos de los problemas que se querrán solucionar, por lo que habrá que crear modelos de redes neuronales compuestos por un número, posiblemente elevado, de ellas. Estas neuronas pueden agruparse dentro de la red neuronal formando capas. Las capas iniciales son las capas de entrada a la red, que contienen los datos sobre los que se trabaja; las finales son las capas de salida, que aportan los resultados obtenidos; y las intermedias son capas ocultas, cuyos valores no son observados durante el entrenamiento. En la Ilustración 17 se aprecia una red neuronal de n entradas, 2 salidas y una capa oculta. El 1 introducido hace referencia a un sesgo, también llamado *bias*, cuyo objetivo es el de conseguir una convergencia más rápida. Cada capa tendrá asignadas sus matrices de pesos y sus vectores de desplazamiento, calculados mediante el entrenamiento. La capa oculta mostrada en la figura tiene las salidas de cada neurona de la capa que le precede conectadas a las entradas de cada una de las suyas. Este tipo de redes se denominan redes totalmente conectadas o *full-connected*.

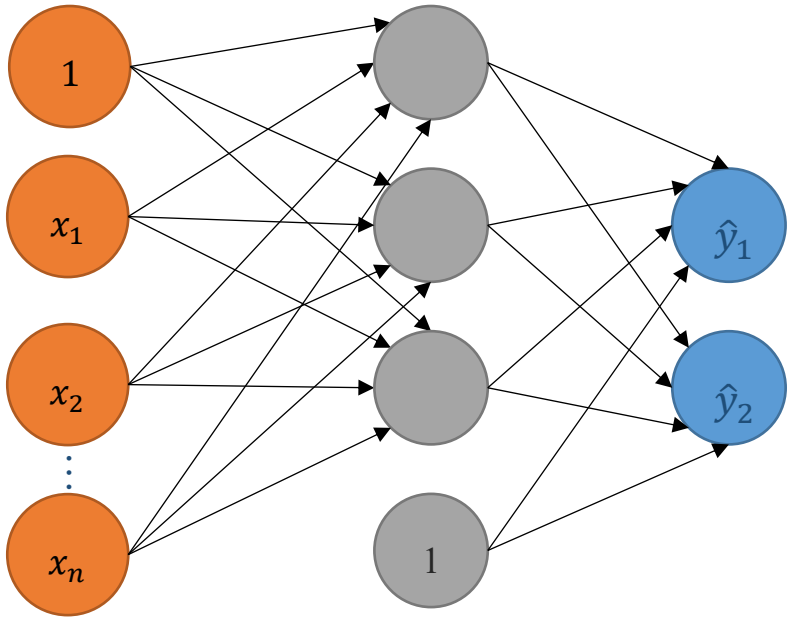

Ilustración 17. Red neuronal de una capa con dos salidas.

La salida de cada neurona de la capa oculta será la evaluación de la función de activación en el punto dado por la suma de las multiplicaciones de cada entrada por el peso asociado a ella y el factor relativo a la *bias*. En la Ecuación 4 se refleja la obtención de dichas salidas, siendo o_i las asociadas a las neuronas i. Los términos x_j se corresponden con cada una de las entradas, w_{ij} es el peso de la neurona i debido a la entrada j y b_i es el factor bias debido a la neurona i.

$$o_1 = f(w_{11}x_1 + w_{12}x_2 + \cdots + w_{1n}x_n + b1)$$
$$o_2 = f(w_{21}x_1 + w_{22}x_2 + \cdots + w_{2n}x_n + b2)$$
$$o_m = f(w_{m1}x_1 + w_{m2}x_2 + \cdots + w_{mn}x_n + bm)$$

Ecuación 4. Salida de una neurona de la capa oculta.

En la Ecuación 5 se modela de igual manera la obtención de las salidas de la red neuronal.

$$\hat{y}_1 = f(w_{11}o_1 + w_{12}o_2 + \cdots + w_{1m}o_m + b1)$$
$$\hat{y}_2 = f(w_{21}o_1 + w_{22}o_2 + \cdots + w_{2m}o_m + b2)$$

Ecuación 5. Salidas de la red neuronal.

Dada la naturaleza de las expresiones, es fácil observar que pueden expresarse los pesos, las entradas y el *bias* como términos matriciales (Ecuación 6). En la Ecuación 7 y en la Ecuación 8 se muestran las ecuaciones que modelan las salidas de las neuronas de la capa oculta y de la red neuronal.

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \ddots & w_{2n} \\ \cdots & \cdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Ecuación 6. Términos matriciales para la obtención de las salidas de la red neuronal.

$$o = f(Wx + b)$$

Ecuación 7. Salida de las neuronas de la capa oculta expresada en términos matriciales.

$$\hat{y} = f(Wx + b)$$

Ecuación 8. Salida de la red expresada en términos matriciales.

3.1.5 Aprendizaje de las redes neuronales

El valor tomado por la matriz de pesos y de bias será obtenido durante el entrenamiento de la red, siendo estos modificados con el objetivo de satisfacer los resultados predichos, tomados a priori, de un conjunto de muestras de entrenamiento. Cuando una red es capaz de resolver problemas complejos se dice que ha aprendido.

Hay diversos métodos de aprendizaje:

- Aprendizaje supervisado:

En el aprendizaje supervisado existe un agente externo (maestro) que determina la respuesta que debe aportar la red en función de las entradas introducidas. Este agente controla la salida, modificando la matriz de pesos y bias cuando esta no es la adecuada, con el objetivo de que en las siguientes iteraciones sea correcta. Para ello es necesario que la red tenga una serie de pares de entrada-objetivo, en los que se indica según el vector de entradas, que vector objetivo debe conseguirse.

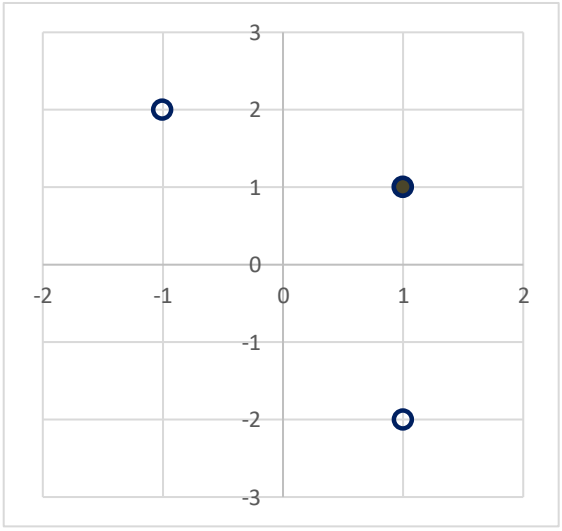

Ilustración 18. Representación de pares entrada-objetivo.

- Aprendizaje reforzado:

En el aprendizaje reforzado se dan las mismas circunstancias que en el aprendizaje supervisado, con la diferencia de que en lugar de haber objetivos vinculados con entradas se tienen notas. En función de si la nota es baja o alta se variará, más o menos, la matriz de pesos y la de bias.

- Aprendizaje no supervisado:

  En el aprendizaje no supervisado no se dispone de un conjunto de ejemplos previamente clasificados con los que entrenar la red. La modificación de la matriz de pesos y de bias viene dada por una categorización de patrones de entrada en un conjunto determinado de clases.

3.1.5.1 Ejemplo de entrenamiento supervisado de un perceptrón

Para una red tipo perceptrón, como el indicado anteriormente, con dos entradas y una salida, y al que se le realiza un entrenamiento basado en aprendizaje supervisado, se pueden tener los siguientes pares de entrada y objetivo (Ecuación 9):

- $\left\{\begin{bmatrix}1\\2\end{bmatrix}\middle|0\right\},\left\{\begin{bmatrix}1\\-2\end{bmatrix}\middle|0\right\},\left\{\begin{bmatrix}1\\1\end{bmatrix}\middle|1\right\}$

Ecuación 9. Pares de entrada-objetivo

Estos se representan en la Ilustración 18 en un gráfico cartesiano, indicando mediante un círculo con relleno el caso positivo (1) y sin relleno el negativo (0). En la Ilustración 19 se muestra el de red neuronal tipo perceptrón que se utilizaría, considerando el parámetro *bias*.

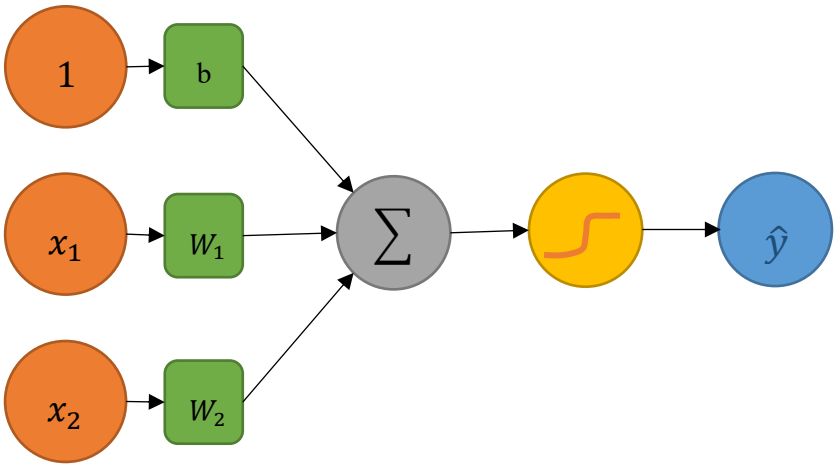

Ilustración 19. Perceptrón dado el caso de aprendizaje.

Si se toma como matriz de pesos inicial aleatoria [1 0.7], como parámetro *bias* 0.3 y como función de activación un escalón se tiene para la entrada $\begin{bmatrix}-1\\2\end{bmatrix}$ el resultado de la Ecuación 10.

$$y = hardlimit\left(0.3 + \begin{bmatrix}1 & 0.7\end{bmatrix}\begin{bmatrix}-1\\2\end{bmatrix}\right) = hardlimit(0.3 - 1 + 1.4) = hardlimit(0.7) = 1$$

Ecuación 10. Solución para pesos iguales a [1 0.7] y bias 0.3 con la entrada [-1; 2].

Se puede observar como los pesos y el bias deben ser modificados ya que la salida encontrada por la red no se corresponde con la dada por el agente externo. En la Ilustración 21 se muestra la frontera obtenida con estos datos, y como para las siguientes entradas el resultado obtenido si se corresponde con el generado por el agente. Sin embargo, al no corresponderse en el primer caso los valores han de ser modificados. Si se utilizasen pesos iguales a $[1 - 0.7]$ y un parámetro *bias* igual a -0.6 se obtendría el resultado de la Ilustración 20 donde todos los elementos consiguen un resultado adecuado. Por lo tanto, estos pesos serían válidos para la red una vez entrenada. Hay que indicar que no únicamente ellos aportarían soluciones que se adecuen al problema, ya que un múltiple conjunto de valores lo harían. Nótese que si el parámetro *bias* fuese igual a 0 la recta que modela la frontera pasaría en toda ocasión por el origen, por lo que para este problema no se encontraría ninguna solución.

El objetivo de la red neuronal es el de buscar una solución para la matriz de pesos y *bias* que minimice el error del conjunto de datos dados.

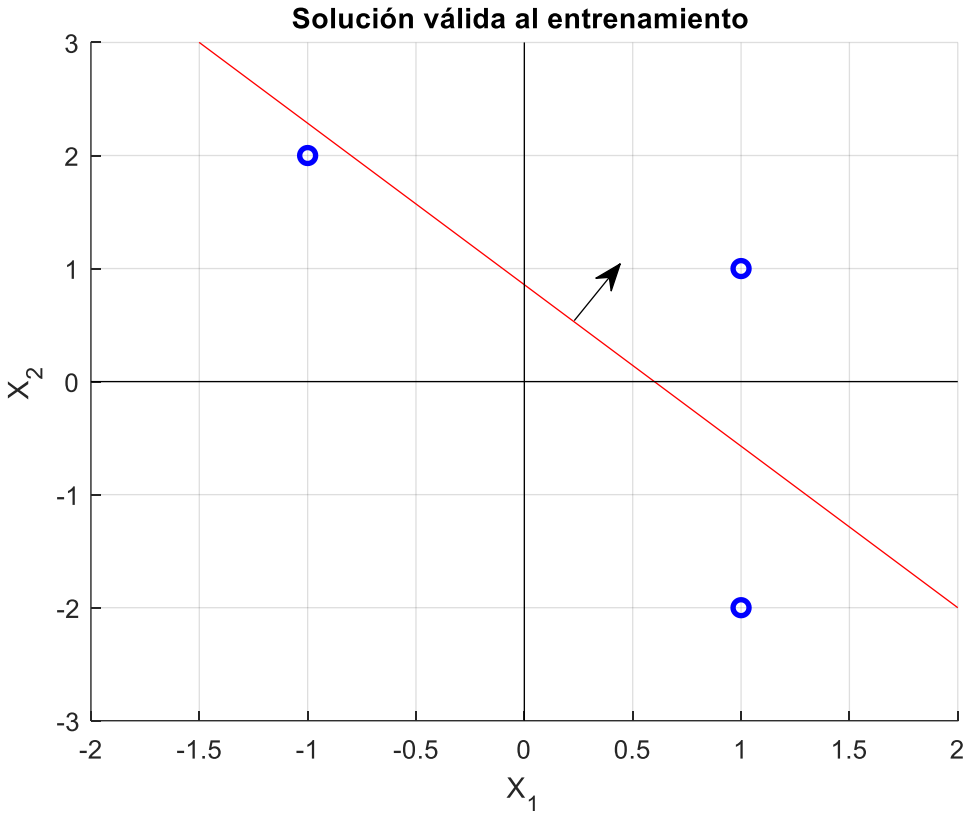

Ilustración 20. Solución obtenida con los pesos [1 -0.7] y el parámetro bias -0.6 para el ejemplo propuesto.

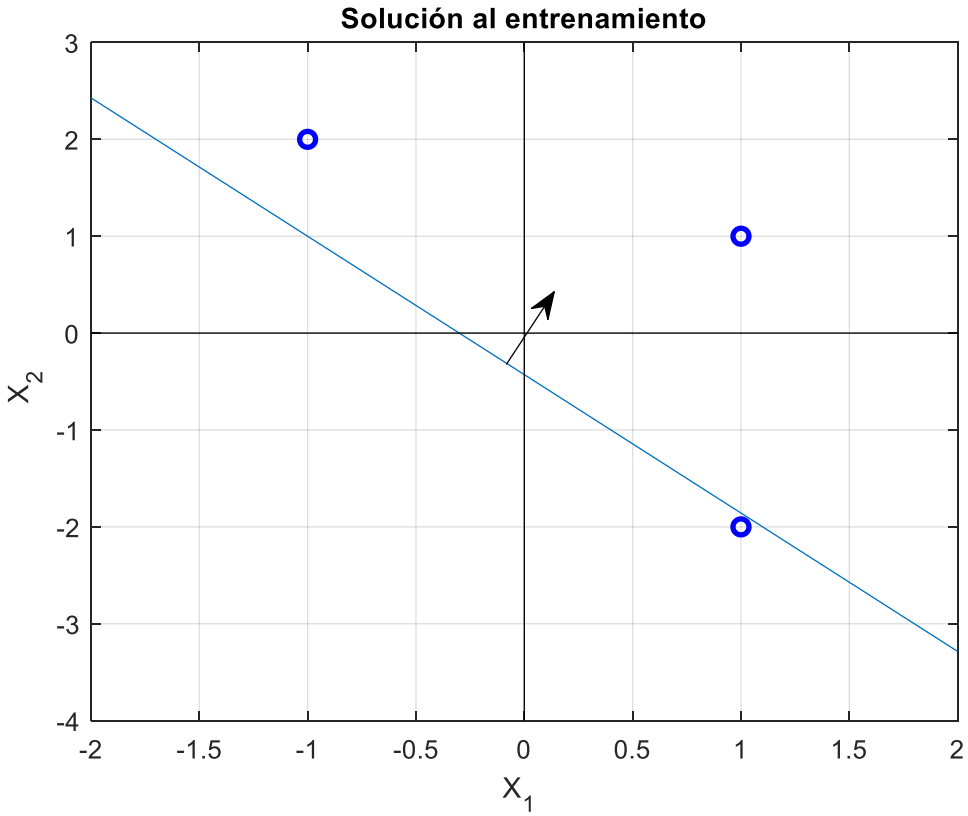

Ilustración 21. Solución obtenida con los pesos [1 0.7] y el parámetro bias 0.3 para el ejemplo propuesto.

.

3.1.5.2 Backpropagation

Una red neuronal artificial, ya considerada como un conjunto mayor a un perceptrón, es entrenada según un algoritmo denominado *Backpropagation* [23] o, en español, retropropagación o propagación hacia atrás de los errores. Este utiliza el algoritmo del Descenso del gradiente, visto más adelante, para buscar aquellos puntos en los que se minimiza la función de coste, dada, por ejemplo, por el error cuadrático medio.

En las redes neuronales, una de las dificultades del ajuste de parámetros es que la modificación de un coste en capas anteriores afecta a las capas posteriores, ya que modifica el flujo de sus conexiones. Además, este también se ve afectado por los pesos dados a las capas posteriores. El algoritmo de propagación hacia atrás de errores permite distribuir dicho error entre las neuronas que han estado involucradas en este y, por tanto, modificar los pesos esquivando dicho problema.

Conceptualmente, su funcionamiento consiste en, una vez obtenidas las salidas generadas por un determinado vector de entradas se comparan con las aportadas por el agente exterior. Así se puede obtener el error para cada una de las salidas de la red. Partiendo ahora de cada una ellas se determina la parte proporcional de error referida a cada una de las neuronas involucradas en esa salida. Tras esto, se pasa a la siguiente capa, en la cual se vuelve a hacer lo mismo, determinar la parte de *culpa* que ha tenido cada una de las entradas para que dicha neurona haya obtenido como salida dicho error, y repartirlo. Cuando se llega a la primera capa, se tienen todos los errores distribuidos, lo que permite modificar los parámetros en función de ello. En la Ilustración 22 se puede ver un ejemplo ilustrado de este proceso. El error generado en la salida $\hat{y}_1$ es repartido, proporcionalmente, entre las entradas a las neuronas que lo han provocado. Igualmente, estos errores repartidos deben pasar a la siguiente capa.

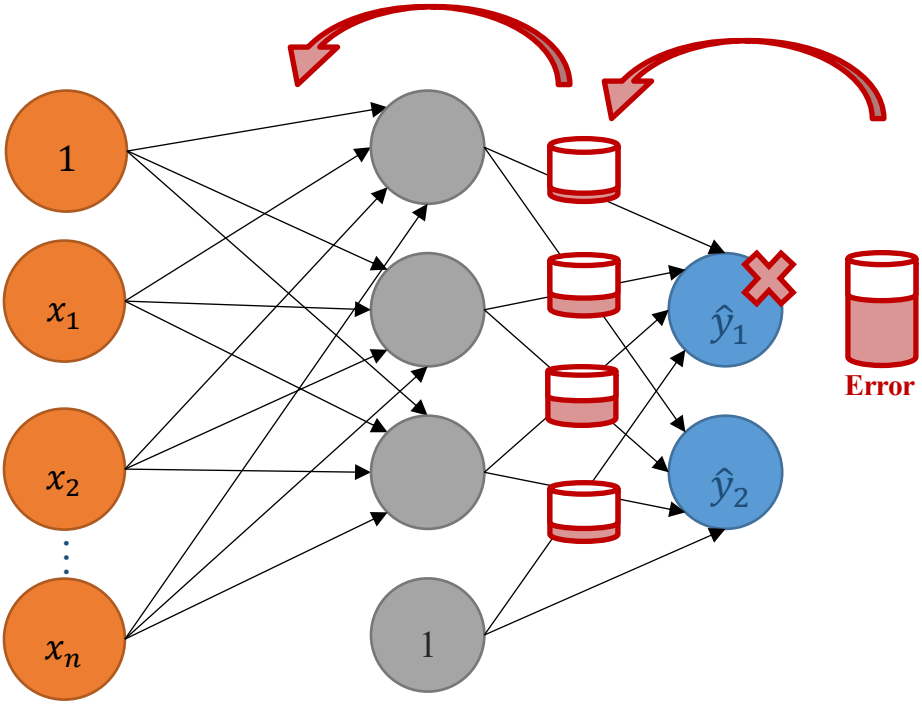

Ilustración 22. Propagación hacia atrás de los errores

Como se pretende usar el descenso del gradiente se necesita conocer lo que cambia el coste con respecto a cada uno de los parámetros de la red, es decir, con respecto a los pesos y al *bias*. Si se tiene en cuenta el último nodo de la red, la ecuación que modela la primera salida se indica en la Ecuación 11, que debe ser además evaluada por la función de coste C.

$$\hat{y}_1 = g(\sum W^m x_i^m + d^m)$$

Ecuación 11. Ecuación que modela la salida y_1 de una red de m capas.

Para obtener las derivadas parciales de dicha función de coste con respecto a w y b se hará uso de la regla de la cadena, como se muestra en la Ecuación 12 y en la Ecuación 13, donde Z^m es igual a $\sum W^m x_i^m + d^m$.

$$\frac{\partial C}{\partial w^m} = \frac{\partial C}{\partial g^m} + \frac{\partial g^m}{\partial z^m} + \frac{\partial z^m}{\partial w^m}$$

Ecuación 12. Derivada del coste con respecto al peso utilizando la regla de la cadena.

$$\frac{\partial C}{\partial b^m} = \frac{\partial C}{\partial g^m} + \frac{\partial g^m}{\partial z^m} + \frac{\partial z^m}{\partial b^m}$$

Ecuación 13. Derivada del coste con respecto al parámetro *bias* utilizando la regla de la cadena.

Las derivadas obtenidas ahora son sencillas de obtener, dependiendo de la función de coste que se utilice, por ejemplo, el error cuadrático medio, y de la función de activación para $\frac{\partial C}{\partial g^m}$ y $\frac{\partial g^m}{\partial z^m}$. La $\frac{\partial z^m}{\partial w^m}$ será igual a la entrada a la red neuronal, es decir, a $g(Z^{m-1})_i$ y la $\frac{\partial z^m}{\partial b^m}$ será una constante igual a 1.

Si se tiene en cuenta la capa anterior *m-1*, la función de coste la indicada es la de la Ecuación 14. Por la regla de la cadena, se tendría que las derivadas del coste con respecto a los parámetros que modelan la red son aquellas indicadas en la Ecuación 15 y en la Ecuación 16.

$$C(y_1) = C(g^m(W^m g^{m-1}(W^{m-1} \cdot g^{m-2} + b^{m-1}) + b^m))$$

Ecuación 14. Función de coste teniendo en cuenta la capa m-1.

$$\frac{\partial C}{\partial w^{m-1}} = \frac{\partial C}{\partial g^m} + \frac{\partial g^m}{\partial z^m} + \frac{\partial z^m}{\partial g^{m-1}} + \frac{\partial g^{m-1}}{\partial z^{m-1}} + \frac{\partial z^{m-1}}{\partial w^{m-1}}$$

Ecuación 15. Derivada del coste con respecto al peso de m-1 utilizando la regla de la cadena.

$$\frac{\partial C}{\partial b^{m-1}} = \frac{\partial C}{\partial g^m} + \frac{\partial g^m}{\partial z^m} + \frac{\partial z^m}{\partial g^{m-1}} + \frac{\partial g^{m-1}}{\partial z^{m-1}} + \frac{\partial z^{m-1}}{\partial b^{m-1}}$$

Ecuación 16. Derivada del coste con respecto al sesgo de m-1 utilizando la regla de la cadena.

Los dos primeros parámetros anteriores se corresponden con los de la Ecuación 12 y la Ecuación 13 y pasarán a denominarse el error dado a la capa L (δ^L). La derivada $\frac{\partial z^m}{\partial g^{m-1}}$ se corresponde con la matriz de pesos y bias de la capa m, $\frac{\partial g^{m-1}}{\partial z^{m-1}}$ es la derivada de la función de activación de la neurona, $\frac{\partial z^{m-1}}{\partial w^{m-1}}$ es la entrada a la neurona $g^{m-2}(Z^{m-2})$ y $\frac{\partial z^{m-1}}{\partial b^{m-1}}$ una constante igual a 1. De esta manera, se distribuye el error de una capa a la anterior. Ahora, el error de la neurona en la capa L-1 (δ^{L-1}) será $\frac{\partial C}{\partial g^m} + \frac{\partial g^m}{\partial z^m} + \frac{\partial z^m}{\partial g^{m-1}} + \frac{\partial g^{m-1}}{\partial z^{m-1}}$, lo que nos permitirá calcular análogamente los costes asociados al resto de capas anteriores.

3.1.5.3 Descenso del gradiente

El descenso del gradiente es un algoritmo de optimización muy utilizado en el campo de las redes neuronales, cuyo objetivo es buscar mínimos en funciones multidimensionales. Este método no puede asegurar encontrar mínimos que sean globales, pudiéndose encontrar una solución perteneciente a uno que sea local. Su funcionamiento consiste en, partiendo de un punto, calcular su gradiente, obteniendo así la dirección de mayor cambio ascendiente, y *caminar* una cierta distancia en aquella dirección que es opuesta, es decir, con mayor pendiente descendiente. El algoritmo convergerá cuando los cambios de gradientes sean cada vez menores o muy similares a 0. En la Ecuación 17 se muestra la expresión que modela este comportamiento. El parámetro η es un ratio de aprendizaje, que indica cuanto se avanza en cada iteración.

Si este es pequeño, el gasto computacional será elevado, dando lugar a una ineficiencia en el algoritmo. Si es grande, las variaciones serán tan grandes que será complicado que este llegue a los mínimos, dando lugar a que el algoritmo no converja. La decisión de este valor es una de las tareas más importantes del modelado de la red.

$$w_{new} = w_{old} - \eta \cdot \nabla f$$

Ecuación 17. Cálculo de un nuevo punto mediante el método del descenso del gradiente.

En la Ilustración 23 se enseña un ejemplo de la búsqueda de un mínimo sobre una superficie. Se comienza desde un punto inicial, y se va caminando sobre la superficie en aquella dirección contraria a la que se produce un mayor cambio ascendente, o lo que es lo mismo, a aquella indicada por el gradiente en ese punto.

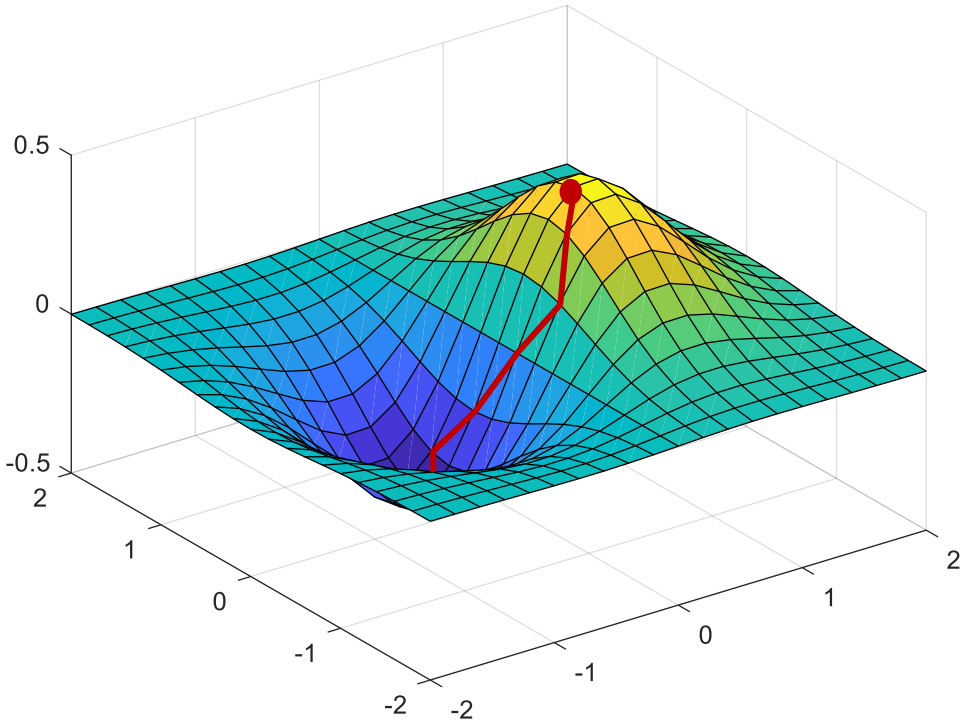

Ilustración 23. Ejemplo del algoritmo de descenso del gradiente.

3.1.6 Sobre entrenamiento y falta de entrenamiento

Cuando se entrena una red neuronal el objetivo que se tiene es que esta llegue a generalizar, es decir, que no solo sepa dar una respuesta certera a aquellas entradas que ya han sido introducidas, sino que haya aprendido a resolver también aquellas que nunca probó, y de las cuales no tiene respuesta. Es en este punto en el que aparecen los conceptos de sobre entrenamiento (*overfitting*) y falta de entrenamiento (*underfitting*).

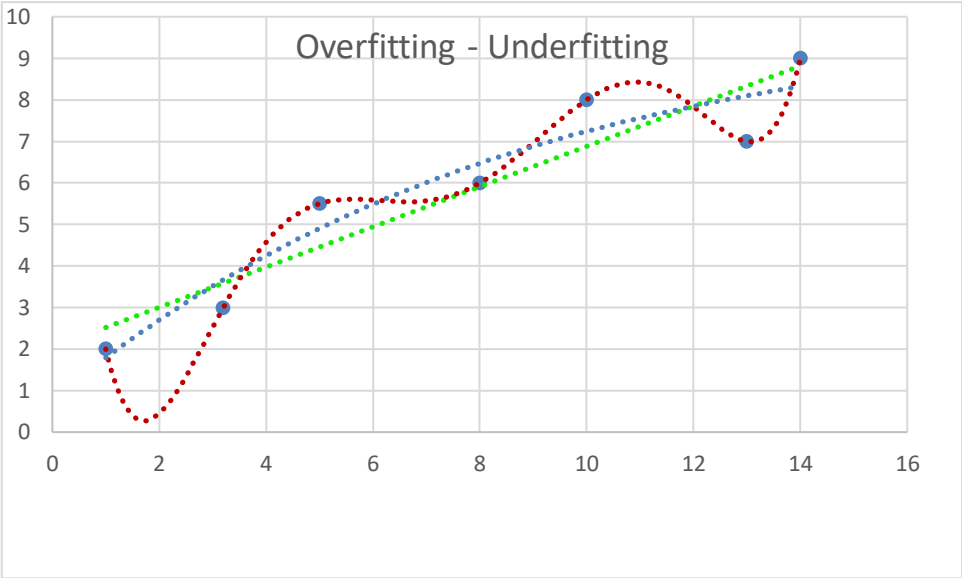

Ilustración 24. Overffitting (rojo) underfitting (verde) y entrenamiento adecuado (azul)

La falta de entrenamiento se da cuando el modelo no se ajusta lo suficientemente bien a los datos introducidos como para que este llegue a generalizar. Por ejemplo, en la Ilustración 24 se puede ver en verde un modelo que no está bien entrenado, ya que para unos datos que dibujan una curva se ha utilizado una solución lineal, no lo suficientemente buena. Sin embargo, para una curva polinómica de orden 3, en azul en la imagen, se muestra un resultado más acorde a los datos, que podría dar soluciones mucho más adecuadas, llegando a generalizar. Al aumentar el orden del polinomio, en rojo en la imagen para uno de orden 6, se obtienen curvas que se ajustan cada vez más a los datos. Sin embargo, esto no implica que este último sea bueno, ya que la red ha llegado a aprender de los datos de entrenamiento, de tal manera que no generaliza para el resto. A esto es a lo que se le denomina *overfitting*, en el cual se produce un modelado del ruido. Uno de los problemas del entrenamiento de las redes neuronales es identificar el punto adecuado de entrenamiento, de manera que no se produzcan ninguno de estos dos casos.

Para conocer si hay sobre entrenamiento se dividen los pares de entrada y salida de manera que se tenga un conjunto de muestras de entrenamiento (≈ 80%), con el que se entrenará la red tal y como se ha indicado, y otro de pruebas (≈ 20%), con el que simultáneamente se probará su adecuado funcionamiento. Es importante que estos datos estén bien distribuidos, de forma que datos semejantes no se encuentren solo en el primer conjunto o viceversa. En ocasiones, puede ser buena idea ordenarlos aleatoriamente, aunque esto no siempre es adecuado. Durante el entrenamiento, como se muestra en la Ilustración 25, el error generado por las muestras de entrenamiento y de las de prueba decae, siendo generalmente mayor el de estas últimas, de forma más abrupta en los primeros momentos y menos al final. Llega un momento, señalizado por un punto naranja en la imagen, en el cual el error de las muestras de entrenamiento sigue cayendo, mientras que el de las de prueba comienza a aumentar. Es en este punto cuando se considera que la red está realizando *overfitting*, ya que para nuevas muestras deja de ofrecer resultados adecuados por haber aprendido de aquellas de entrenamiento. Un buen resultado de entrenamiento de la red se va a encontrar, por tanto, en dicho instante.

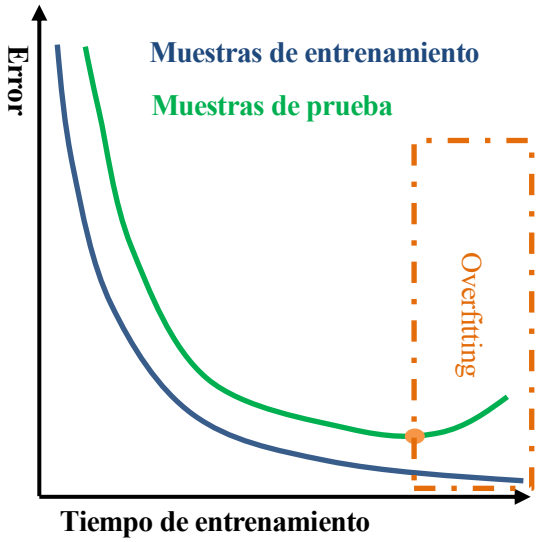

Ilustración 25. Evolución típica del error de entrenamiento y prueba.

3.2 Redes Neuronales Convolucionales (CNN – Convolutional Neural Networks)

Las redes neuronales convolucionales o *convolutional neural networks (CNNs)*, en inglés, son redes neuronales artificiales que intentan asemejar el córtex visual mamífero. En [24] son entrenadas con una *GPU* para la clasificación de imágenes, campo en el que son frecuentemente usadas por requerir una densidad de red menor, comprobándose los buenos resultados obtenidos. Su principal diferencia con respecto al perceptrón multicapa, anteriormente visto, es que está formada por capas convolucionales y de submuestreo alternadas, terminando con un conjunto de capas full-connected. A continuación, se van a comentar estas fases.

Ilustración 26. Conversión de matriz a vector.

3.2.1 Preprocesado de los datos

En la actualidad, por lo general las imágenes están compuestas por conjunto de pixeles de 8 bits, lo que implica un rango de valores entre 0 y 255. En las redes neuronales estos datos suelen normalizarse, ya sea dividiendo entre 255, en cuyo caso el nuevo rango de valores se encontraría entre 0 y 1 o realizando una substracción de la media, donde se obtiene un rango entre -1 y 1. Esto se realiza por motivos computacionales, ya que mejora la velocidad de convergencia y evita, en ocasiones, problemas con gradientes, que provocan pesos iguales a 0 o que tienden a infinito.

Además, en muchas ocasiones se modifica la estructura de la matriz que determina la imagen, convirtiéndola en un vector, como se muestra en la Ilustración 26.

3.2.2 Capa de convolución

En la capa de convolución, con el objetivo de disminuir el tiempo de convergencia de la red, las entradas, es decir, el acceso al valor tratado de los pixeles, no están todas conectadas con las neuronas pertenecientes a la capa oculta. En ella, como su nombre indica, se realizan las convoluciones entre la imagen y un filtro o *kernel*, modelado por la matriz de parámetros, y que inicialmente tomará valores aleatorios, siendo estos modificados según el algoritmo de *backpropagation*. Estas consisten en productos y sumas entre dos conjuntos de datos.

En la Ilustración 27 se realiza una convolución entre la primera matriz, que compone los datos de entrada, y un filtro de Sobel, muy utilizado en visión por computador para la detección de bordes. Esta es efectuada utilizando un factor de salto en x e y (S_x, S_y) igual a 1, es decir, se hará un recorrido de un espacio en x cada vez que se realice un producto, y una vez no se pueda avanzar más, se realizará el salto en y, volviendo al inicio en x.

0.3	0	0.1	0.2	0
0.2	0	0	0.1	0.1
0.4	0.1	0.2	0.1	0.2
0.6	0.2	0	0	0.5
0.5	0	0	0.5	0.6

$*$

1	0	-1
2	0	-2
1	0	-1

$=$

0.8	-0.4	-0.1
1.2	0.1	-0.6
1.9	-0.1	-1.6

Ilustración 27. Ejemplo de convolución.

Para el primero, centrado en la posición $x = 2$ e $y = 2$ se tienen el siguiente resultado:

$$0.3 * 1 + 0.1 * -1 + 0.2 * 2 + 0 * -2 + 0.4 * 1 + 0.2 * -1 = 0.8$$

La dimensión de la capa de la red neuronal dependerá del tamaño de la entrada, del tamaño del filtro, del factor de salto y del número de filtros que se empleen. Por ejemplo, para *MTCNN*, que se verá más adelante, su primera red tiene como entrada una imagen RGB de tamaño $12x12$ y se utilizan 10 filtros de tamaño $3x3$, lo que da como resultado una salida de $10x10x10$.

A esta convolución debe de aplicársele un desplazamiento y la función de activación, que para las *CNN* se suele ser de tipo *ReLU*, aunque para el caso de *MTCNN* se utiliza la *PreLU*, similar a esta última con la diferencia de que para valores anteriores a 0 su valor no es nulo, manteniéndose en una recta, como la dibujada en la Ilustración 28.

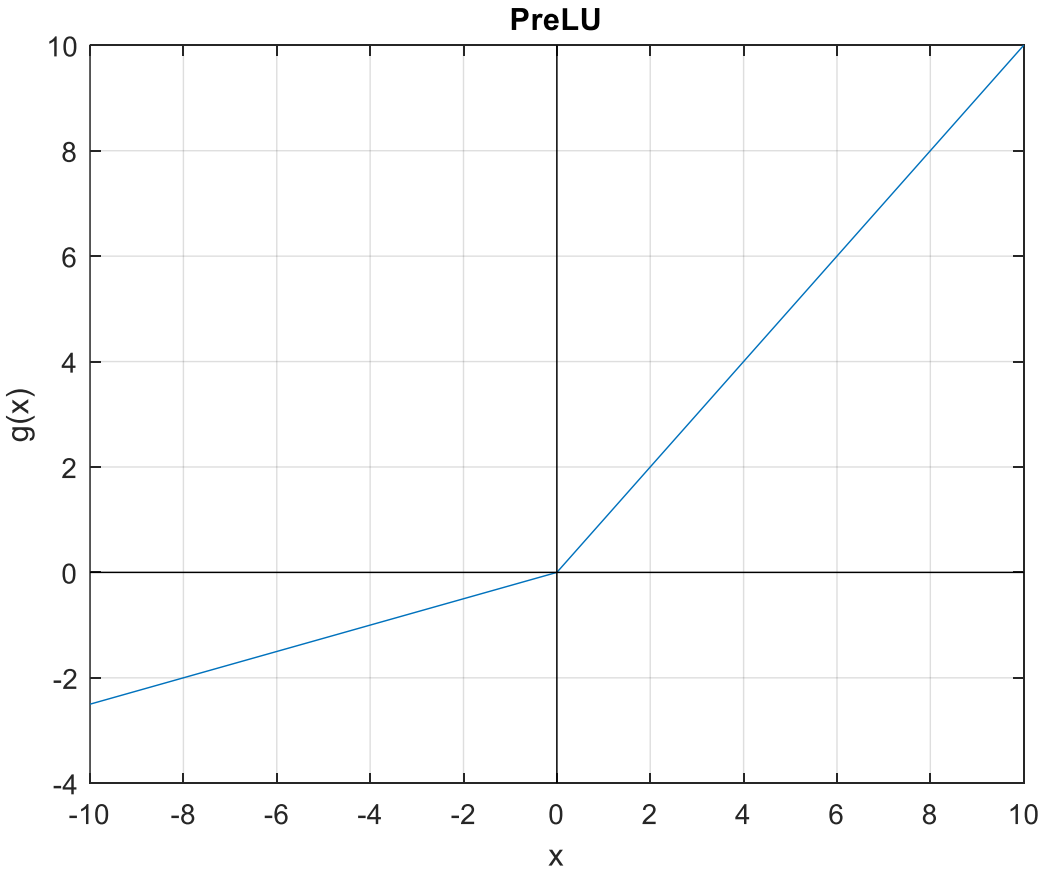

Ilustración 28. Función de activación PreLU.

3.2.3 Submuestreo con *Max-Pooling*

Si se continua con el caso de la red *MTCNN*, se puede inferir que para una imagen de tamaño muy reducido ($12x12x3$ pixeles) se necesita un número elevado de neuronas en la capa de entrada (432 neuronas). Si, además, como se mostró más atrás, se utilizan 10 filtros de dimensiones $3x3$, tras la convolución de esta primera capa se obtienen 10^3 salidas. Cuando el tamaño de la imagen aumenta también lo hace la densidad de la capa, provocando que el gasto de cómputo sea muy alto. Es por ese motivo que se utiliza una capa de submuestreo en las primeras capas de la red. Para ello hay distintos métodos, pero el más conocido es el *Max-Pooling*, que consiste en pasar un filtro, de determinadas dimensiones y factor de salto, cuyo resultado será el valor más alto de todos los datos que comprenda. De esta manera, se queda con aquellas características que son más importantes. Por ejemplo, en la Ilustración 29 se utiliza un kernel de tamaño $2x2$ y desplazamiento 2. La primera iteración que realiza abarca la matriz $\begin{bmatrix}0 & 1\\0 & 1\end{bmatrix}$ cuyo valor máximo es 1, dando lugar a que el primer dato de la salida sea un 1. Al producirse el desplazamiento se enfoca en $\begin{bmatrix}0 & 0\\1 & 0\end{bmatrix}$, ocasionando un 1 en la salida. Si se continua, se obtiene el resultado mostrado.

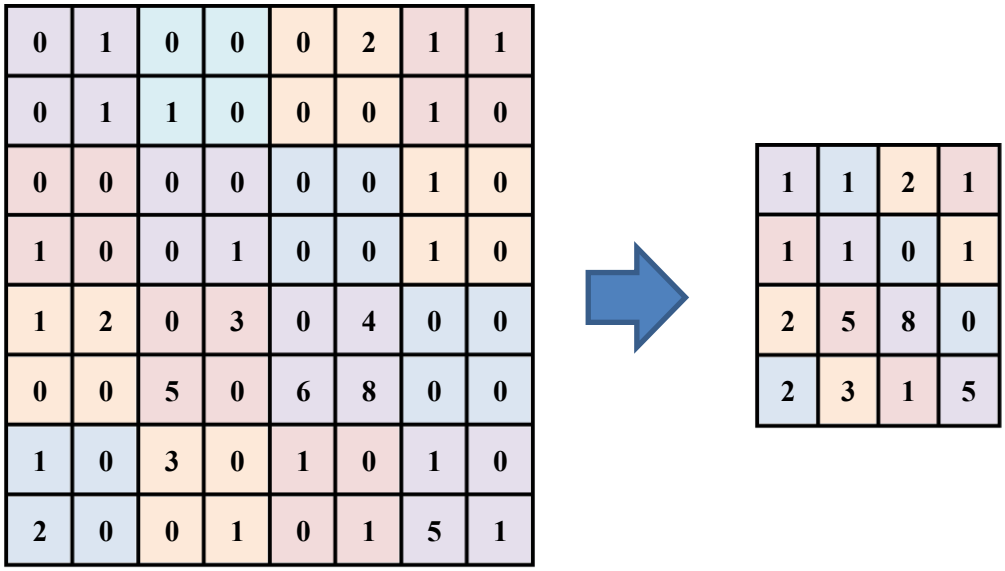

Ilustración 29. Efecto de realizar la operación de Pooling con una kernel de tamaño 2x2 y desplazamiento igual a 2.

Con esta técnica se ha pasado de una matriz de dimensiones $8x8$ a otra de $4x4$, minimizando el coste de cómputo. Para la primera etapa de *MTCNN* se había comentado que se obtenía una salida en la convolución de tamaño $10x10x10$. Al aplicar el submuestreo de la Ilustración 29 este se disminuye a $5x5x10$.

3.2.4 Capa Full-Connected

La capa Full-Connected es una capa densamente conectada, con todas sus entradas vinculadas con los parámetros de la red, que suele encontrarse en las últimas etapas de una *CNN*. Su principal función es como clasificador, es decir, determina la clase a la que pertenece la entrada introducida una vez tiene la salida de las capas que le preceden. En la Ilustración 17 se muestra este tipo de capas.

3.2.5 Esquema de una red neuronal convolucional

Como se ha ido comentando en los apartados anteriores, una red neuronal convolucional está compuesta por diversas capas. Hay que indicar, que en ocasiones se toman como diferentes aquellas que son de convolución y de submuestreo, sin embargo, para este proyecto se considerarán como una propia, por no tener, esta última, parámetros de peso o sesgo. Por lo tanto, se puede decir que en una red neuronal se parte de una entrada, que por lo general debe ser preprocesada, y se pasa a un número determinado de capas de convolución, que a su vez tienen, o no, asociado un submuestreo, probablemente según el algoritmo de *Max Pooling*. Finalmente, una vez ha obtenido un vector de características asociado a la entrada, gracias a una capa *Fully Connected* se obtiene la salida de la red. En la Ilustración 30 puede verse ejemplificado este proceso. Hay que indicar, que la salida aportará un valor indicativo de la confianza que tiene la red de que dicha entrada pertenezca a la clase determinada por ella misma.

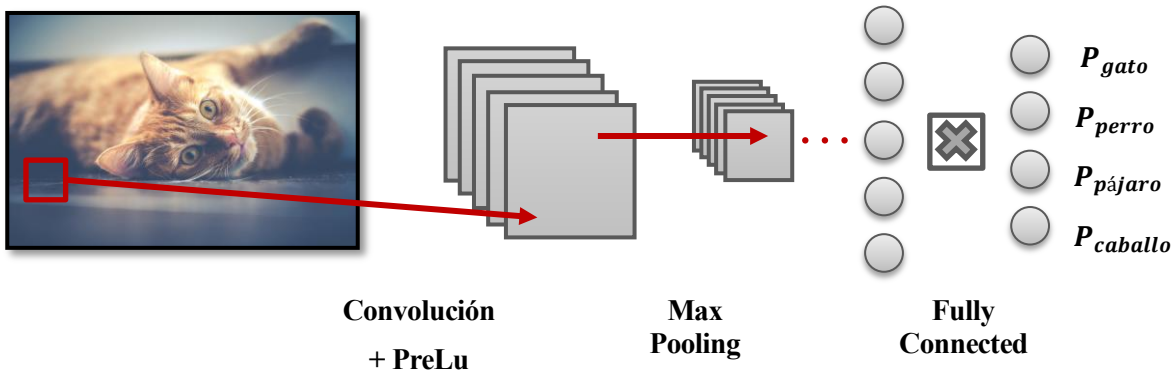

Ilustración 30. Estructura de una red neuronal convolucional.

4 DETECCIÓN DE ROSTROS

Visualizo una época en la que (los humanos) seremos a los robots lo que los perros son para nosotros.

Claude Shannon

Para obtener descriptores de un rostro que nos permitan su identificación es inicialmente necesario obtener la zona en la que este se encuentra dentro de la imagen. Para ello, existen diversas técnicas, comentadas previamente en el apartado 1.2. Hoy en día, la mayoría de los enfoques se basan en el uso de redes neuronales convolucionales, que obtienen buenos resultados incluso ante situaciones de cambios de iluminación, oclusión, caras no dispuestas frontalmente o gesticulación.

En este proyecto se van a comparar dos técnicas, por un lado, una propuesta por *DLIB*, y proporcionada por *4i*, que se basa en el uso de *HOG* junto con *SVM*; por otro, *MTCNN*, que hace uso de redes neuronales convolucionales. Los archivos *network.cpp* y *network.h*, mencionados en el apartado 1.3, se encargan de la implementación de esta red neuronal.

4.1 DLIB

DLIB ofrece un conjunto de herramientas implementadas en el lenguaje de programación C++ con algoritmos de aprendizaje automático como redes neuronales, máquinas de vectores de soporte (*SVM*) o algoritmos de agrupamiento, entre el que se encuentra *k-means*. Ofrece una licencia de código abierto, o en inglés *Open Source*, para cualquier tipo de aplicación, utilizándose en multitud de campos, ya sea de tipo industrial, comercial o académico. En la Ilustración 31 aparece el logo de *DLIB*.

Ilustración 31.Logo de DLIB.

Para la detección de rostros DLIB ofrece varias soluciones. Por ejemplo, se implementan clasificadores en cascada AdaBoost junto con las características de Haar, técnica basada en [2]. Otro método se orienta hacia el aprendizaje profundo, empleando para ello una red neuronal convolucional (*CNN*). Finalmente, también se implementa un algoritmo que hace uso de *HOG* (*Histogram of Oriented Gradients*) como descriptor junto con *SVM* (*Support Vector Machine*). Este último ha sido empleado en el proyecto, siendo el programa tomado de la empresa *4i* para compararlo con el detector y alineador de rostros *MTCNN*. A continuación, se va a comentar de manera superficial en que consiste el descriptor *HOG* y el clasificador *SVM*.

0	0	0
-1	1	0
0	0	0

a) Px

0	-1	0
0	1	0
0	0	0

b) Py

Ilustración 32. Plantillas para el cálculo del gradiente en las direcciones X e Y

El histograma de gradientes orientados (*HOG*) es un descriptor de características utilizado en visión por computador para la clasificación de texturas y la detección de objetos y personas, entre otros. Para su obtención se divide la imagen en celdas, de tamaño determinado. Para cada una de ellas se calcula el histograma de gradientes, que representa la distribución del módulo dentro de la misma según su orientación, calculada en el rango 0º-180º. Con el objetivo de que sea invariante a cambios de iluminación, el histograma es normalizado. La agrupación de los valores obtenidos genera un vector de características, denominado descriptor, que modela el elemento presente en la imagen. Para representarlo se utilizan los valores más significativos calculados en la celda, como puede verse en la Ilustración 33.

Ilustración 33. Representación de HOG.

Una máquina de vectores de soporte (*SVM*) es un algoritmo de clasificación lineal cuya frontera de decisión está delimitada por un hiperplano. Cuando se tienen dos clases linealmente separables, donde el número de hiperplanos que las divide es infinito, en *SVM* la solución generada se corresponde con aquella que

maximiza la distancia entre los hiperplanos que contienen los vectores de soporte. Estos últimos, se corresponden con una serie de muestras de entrenamiento que se encuentran en la frontera. En la Ilustración 34 se muestra en la imagen a) un ejemplo de ello, donde existen dos clases, círculos y triángulos, enfrentadas en función de dos características. Los vectores de soporte ayudan a aportar todas las posibles soluciones al problema, y la maximización de la distancia genera el hiperplano solución a nuestro problema. Sin embargo, la mayoría de los casos que vamos a encontrarnos ya sea por la existencia de ruido en las medidas o incluso por categorizaciones incorrectas en las muestras de entrenamiento, no van a dar lugar a lo que se denomina separabilidad entre clases.

Un ejemplo de ello se puede ver en el caso b) de la Ilustración 34. Como solución, *SVM* emplea dos métodos, por un lado, lo que se denomina *margen suave*, que consiste en relajar el grado de separabilidad, consintiendo la existencia de errores en la clasificación de las muestras de entrenamiento. Por otro lado, utiliza lo que se denomina *kernel trick*, que se basa en la transformación del espacio de características en otro de dimensión superior donde sí que exista separabilidad entre las clases.

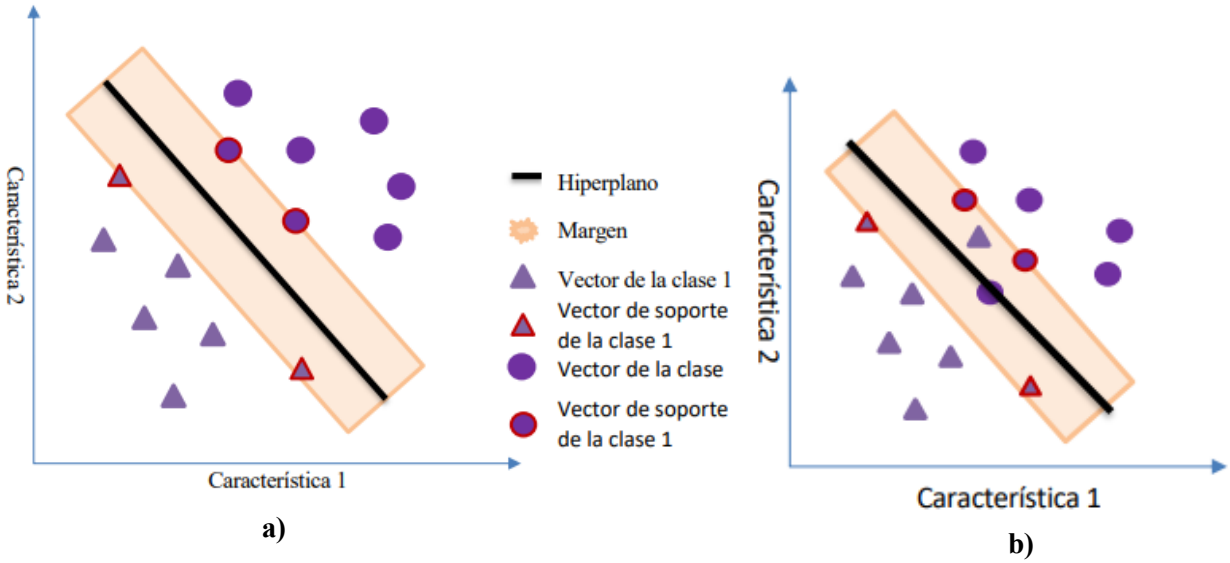

Ilustración 34. SVM. a) Separación entre clases. b) Muestras de entrenamiento en el margen.

Las agrupaciones de ambos algoritmos dan lugar al detector de rostros empleado en DLIB. Su problema, como se verá más adelante, se basa principalmente en la dificultad que presenta ante rostros que no son frontales o con oclusiones. En la Ilustración 35 se muestra el resultado de detectar rostros mediante este método.

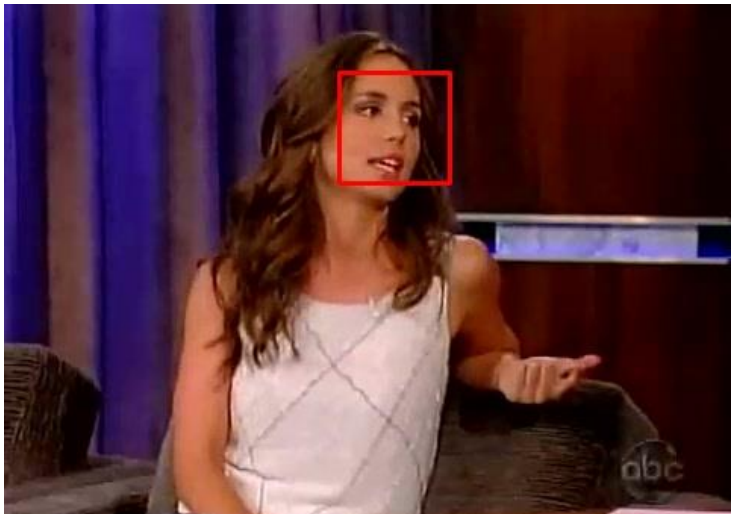

Ilustración 35. Detección de rostros utilizando el detector proporcionado por DLIB.

4.2 MTCNN

MTCNN (*Multi-Task Cascaded Convolutional Networks*), [7], es un algoritmo basado en redes neuronales convolucionales que trata de realizar conjuntamente las tareas de detección de rostros y alineación. Para ello, este algoritmo utiliza tres etapas. En la primera se pretende obtener, de una forma rápida, posibles candidatos en la imagen. En la segunda se mejoran los resultados obtenidos eliminado ventanas que no contienen caras. Finalmente, en la tercera se refina el resultado y se obtienen 5 marcas características. Para ello, cada una utiliza una CNN (Convolutional Neural Network) diferente y más compleja, denominadas respectivamente *Proposal Network (P-Net)*, *Refine Network (R-Net)* y *Output Network (O-Net)*. Antes de introducir datos para la primera etapa se redimensiona la imagen creando una pirámide, como la de la Ilustración 36, en la que, como mínimo su tamaño será igual a $12x12$ pixeles.

Ilustración 36. Pirámide obtenida con la función PyrDown de OpenCV.

4.2.1 Proposal Network (P-Net)

P-Net es una red neuronal convolucional utilizada en la primera etapa de *MTCNN* con el objetivo de obtener los primeros posibles candidatos a rostros en la imagen. En la Ilustración 38 se muestra el esquema de su arquitectura.

La entrada a *Proposal Network* es una ventana de tamaño $12x12x3$, correspondiente a los canales *Red*, *Green* y *Blue* de una porción de imagen, como la que se muestra en verde en la Ilustración 37. Las ventanas serán adquiridas al recorrer cada una de las imágenes que se obtuvieron previamente con el escalado de la original. El objetivo de la red es proporcionar un valor de confianza que nos permita determinar si alguna de estas contiene un rostro.

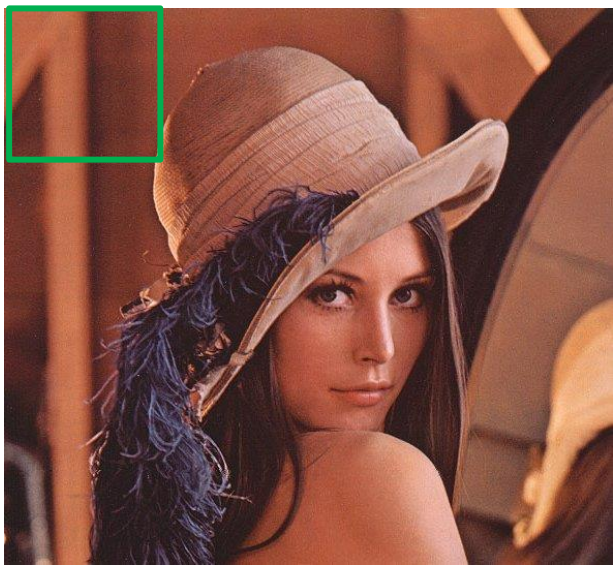

Ilustración 37. Representación de una ventana de entrada a P-Net.

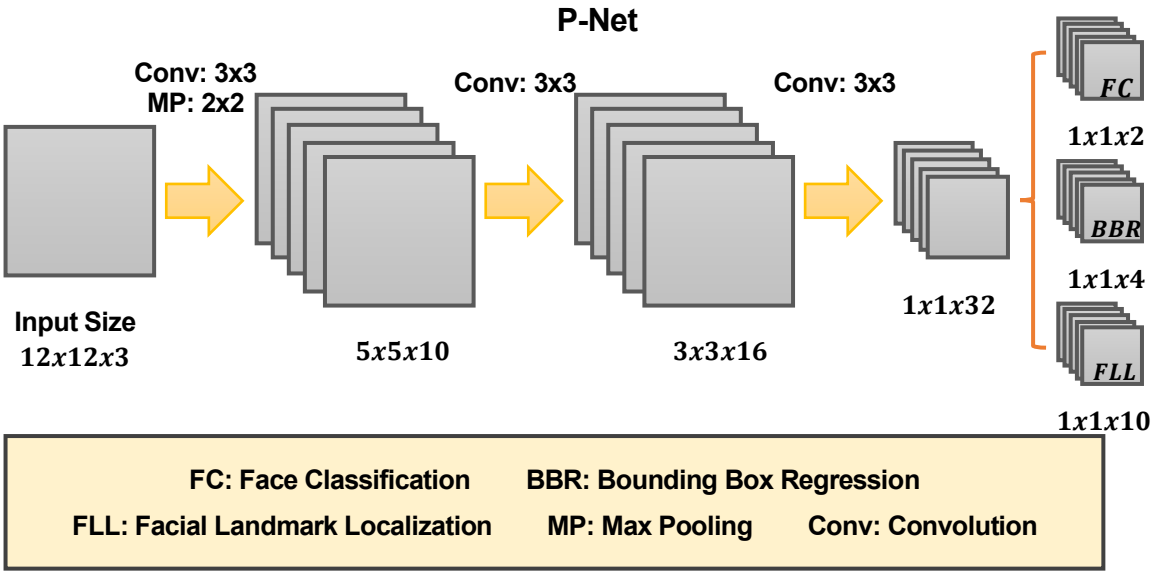

Ilustración 38. P-Net. Representación obtenida a partir de "Joint Face Detection and Alignment using MTCCN"

Antes de realizar operaciones sobre los datos estos deben ser tratados, de manera que se realiza una substracción de la media y un escalado, obteniendo como resultado una matriz con valores en el rango [-1, 1].

Consiguientemente, para cada una de las entradas se realiza una convolución con filtros de tamaño $3x3$, tras lo cual se aplica un desplazamiento (*bias*) y la función de activación *PReLU*, donde los valores menores que 0 serán multiplicados por un número α, adquirido previamente mediante el entrenamiento de la red, y el resto, se mantendrán.

Tras ello, se realizará una capa de agrupación (*pooling*) en la cual se reduce el número de valores que se han obtenido. Para ello, se utiliza un filtro de dimensión $2x2$ que se desplaza según un offset igual a 2. De los 4 valores de la imagen contenidos en el filtro se quedará con aquel que sea mayor. Por ejemplo, en la Ilustración 29 se aplica sobre un mapa de valores de tamaño igual a $8x8$, y se obtiene otro de $4x4$.

Tras ello se realizan otras dos capas de convolución, seguidas cada una de ellas de la aplicación de un desplazamiento y una función *PReLu*. Una vez se ha llegado a este punto la red se divide en dos capas. En la primera se añade la función *Softmax*, que permite obtener la probabilidad de que haya un rostro, o no, en dicha ventana; mientras que la segunda realiza una regresión de los cuadros contenedores, obteniendo así sus coordenadas. En la Ilustración 39 se representa el diagrama de funcionamiento que se acaba de mencionar.

Los valores de los pesos con los cuales se realiza la convolución, de los desplazamientos (*Bias)* y del parámetro α de la función PReLu se adquieren mediante el entrenamiento de la red.

Ilustración 39. P-Net: Diagrama.

4.2.2 Refine Network (R-Net)

R-Net es una red neuronal convolucional utilizada en la segunda etapa de *MTCNN* con el objetivo de refinar los resultados aportados por *P-Net*. En la Ilustración 40 se muestra un esquema de su arquitectura.

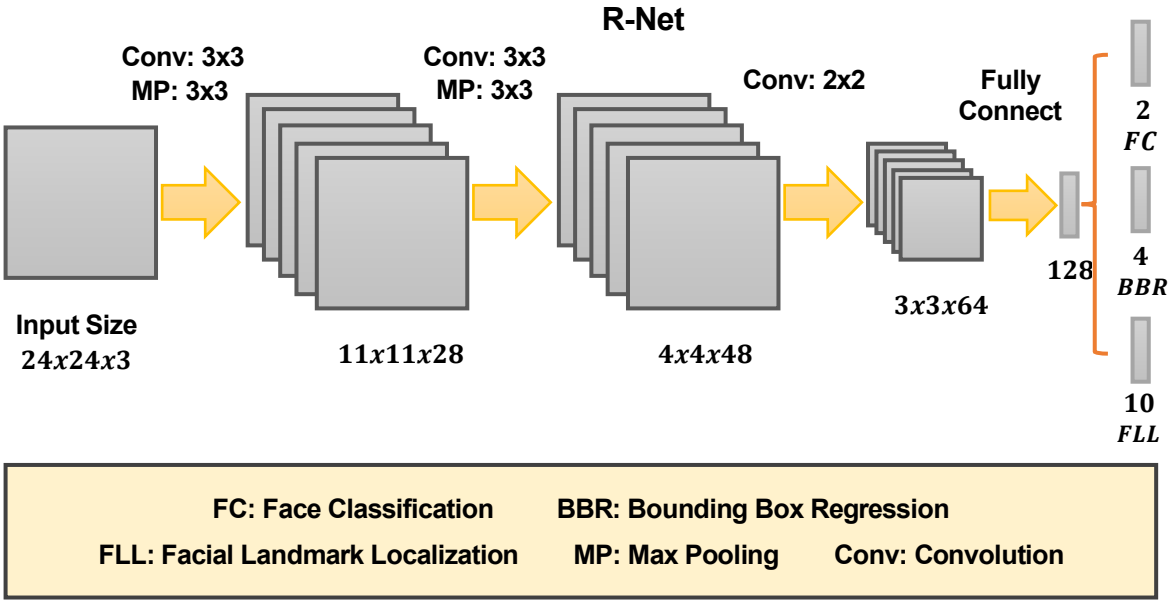

Ilustración 40. R-Net. Representación obtenida a partir de "Joint Face Detection and Alignment using MTCCN"

La entrada a esta red son imágenes de $24x24x3$ pixeles que contengan cada uno de los posibles candidatos encontrados en la etapa anterior. Para ello, se deben dimensionar las ventanas contenedoras. Sobre estas se realizan dos etapas de convolución, con un filtro de tamaño $3x3$, seguidas de la función *PreLu*, un desplazamiento (*Bias*) y de una capa de agrupación con un filtro de $3x3$ pixeles. Con ellas, como salida se obtienen conjuntos de datos de tamaño $11x11x28$ y $4x4x48$ respectivamente. Posteriormente se hace una convolución con un filtro de $2x2$, esta vez seguida tan solo de una función *PreLu* y el desplazamiento, y una *Fully Connect* (convolución con todas las entradas y salidas conectadas entre sí). Tras esta última, se tienen vectores con 128 características, que tras convoluciones *Fully Connect*, desplazamientos y una función *SoftMax* para el primero de los casos son convertidos en vectores de 2 y 4 características con la confianza y la localización. Solo aquellas ventanas que hayan obtenido puntuación superior a un umbral, igual a 0.7 en este c aso, pasan a la siguiente etapa. En la Ilustración 41 se muestra un diagrama con las etapas relativas a esta fase.

Ilustración 41. R-Net: Diagrama.

4.2.3 Output Network (O-Net)

O-Net es una red neuronal convolucional utilizada en la tercera etapa de *MTCNN* con el objetivo de obtener la salida de *MTCNN*, refinando más los resultados anteriores obtenidos y consiguiendo 5 puntos característicos del rostro humano. En la Ilustración 42 se muestra un esquema de su arquitectura.

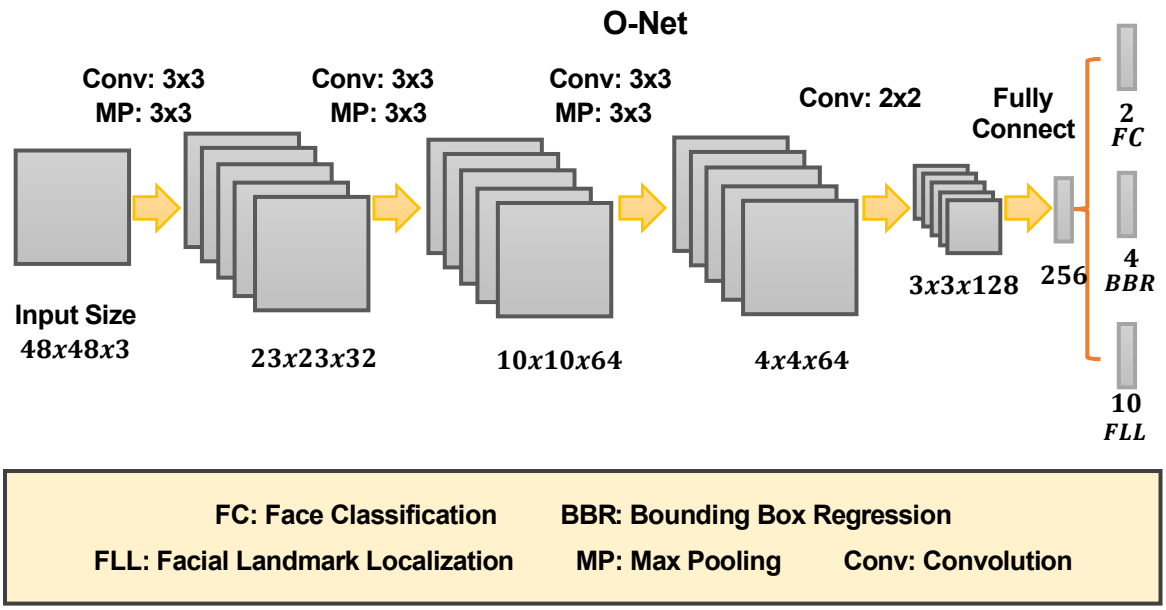

Ilustración 42. O-Net. Representación obtenida a partir de "Joint Face Detection and Alignment using MTCCN"

La entrada es una imagen RGB, relativa a las ventanas contenedoras de rostros obtenidas en la etapa anterior, de tamaño igual a $48x48x3$ pixeles. Está compuesta por 4 convoluciones de tamaño seguidas de un desplazamiento, una función *PreLu* y, a excepción de la última, capas de agrupación (*Pooling*) de tamaño $3x3$. Con ello se obtienen agrupaciones de datos agrupados en matrices de tamaño $3x3x128$. A estos se le aplica una capa *Fully Connect*, con la que se obtienen vectores con 256 características, a las que se le aplican separadamente 3 capas Fully Connect, dando lugar a vectores de tamaño igual a 2, a 4 y a 10, con la confianza de la ventana, su posición y las coordenadas de los 5 puntos característicos de los rostros encontrados. En la Ilustración 43 se muestra un diagrama con las etapas de esta red.

Ilustración 43. O-Net: Diagrama.

4.2.4 NMS (Non-Maximum Suppression)

En visión por computador, cuando se realizan tareas de detección, ya sea de objetos, personas o animales, es frecuente encontrar varias ventanas que definen la posición en la que se encuentra un mismo elemento dentro de la imagen. A pesar de que ninguna de estas regiones puede considerarse un falso positivo, este resultado no es beneficioso, ya que de él se infiere que sobre dicha representación se han encontrado varios objetos, aun cuando solo hay uno. Para solucionar este problema se utiliza NMS (Non-Maximum Suppression). En la Ilustración 44 puede apreciare este efecto, llegándose a detectar 3 rostros cuando tan solo hay uno en la primera imagen. Sin embargo, en la segunda se encuentra una, ya que se ha aplicado dicho algoritmo.

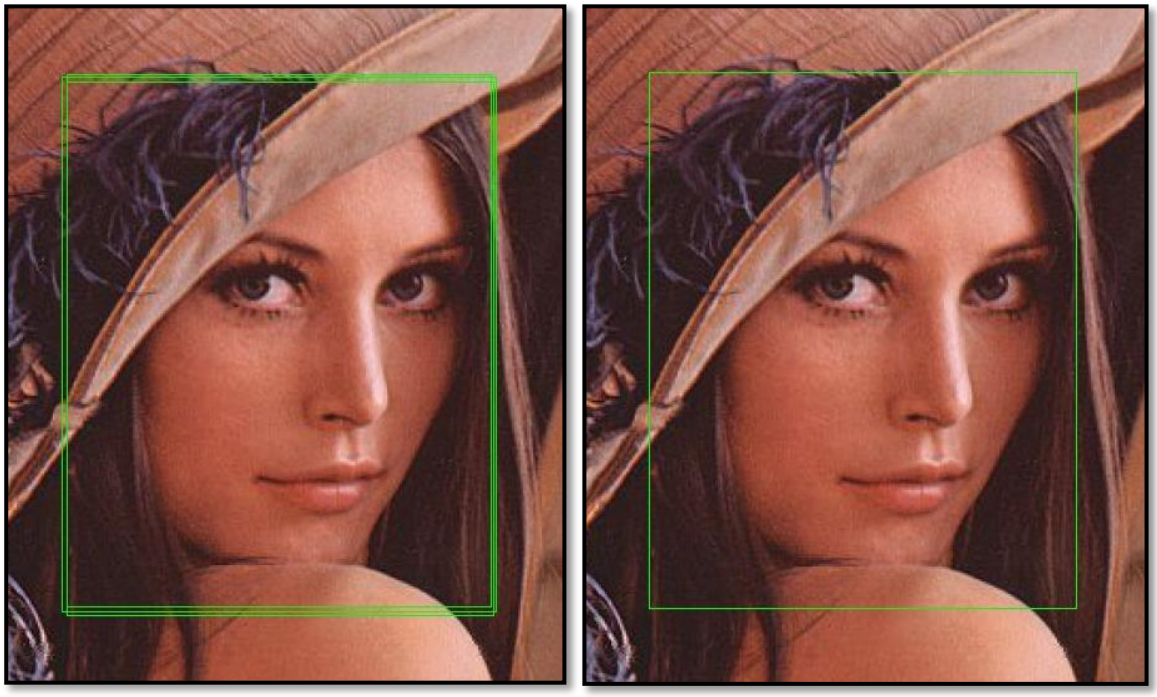

Ilustración 44. Múltiples soluciones a la detección de rostros cuando en la imagen solo se encuentra uno.

En este proyecto, la *NMS* se apoya de la intersección sobre la unión (IoU). La IoU compara el área que se superpone entre dos rectángulos, que se considera contienen el objeto, y la unión de ambos. En la Ilustración 45 se muestra esquemáticamente dicho concepto.

Hay que indicar que esta funcionalidad también posibilita otro método para calcular la *IoU* que consiste en variar el denominador de manera que este sea igual al área menor de ambos rectángulos. Este método se muestra en la Ilustración 46.

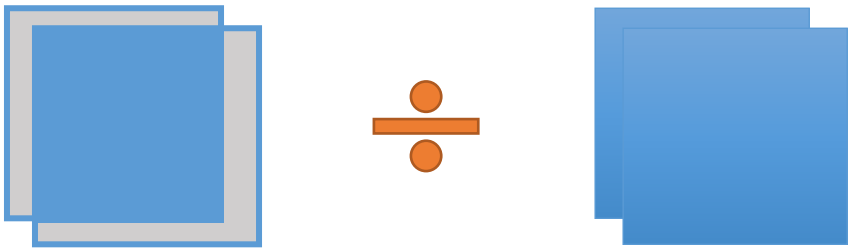

Ilustración 45. Cálculo de la IoU a partir de la unión.

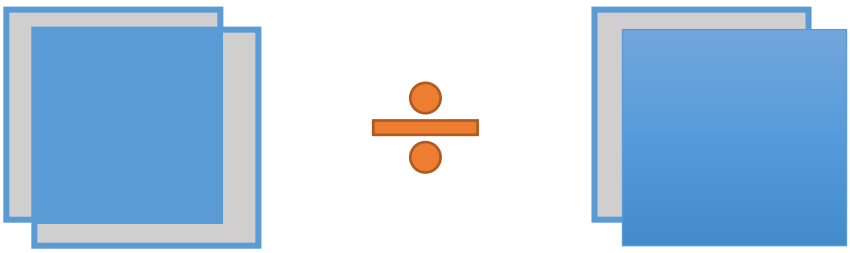

Ilustración 46. Cálculo de la IoU a partir del área máxima

Como las detecciones que hace cada una de las redes neuronales tienen asignada una confianza también nos valemos de esta para determinar aquellas zonas que se consideran como solución. El algoritmo que va a utilizarse ordena, de menor a mayor, los elementos que se han obtenido según su puntuación. Saca los últimos, es decir, aquellos con mayor confianza, y compara si este ha sido ya suprimido. De no haberlo sido, es "eliminado" y comparado con el resto de los cuadros. Para ello, se calcula el rectángulo superpuesto entre ambos para obtener la IoU. En caso de ser esta mayor a un valor umbral, lo que indica que ambas contienen el mismo objeto o señalan el mismo espacio, la comparada es eliminada. Cuando se han tratado todos los rectángulos se puede considerar que los comparados son aquellos que contienen el objeto. En la Ilustración 47 se muestra el diagrama que modela dicho algoritmo.

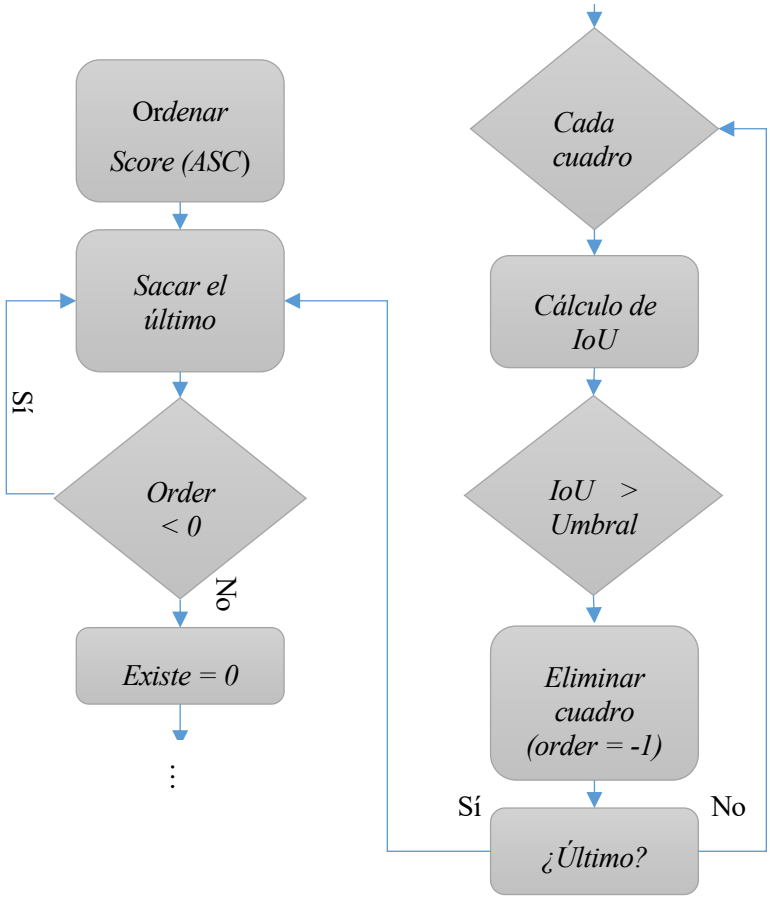

Ilustración 47. Diagrama NMS.

4.2.5 Funcionamiento de MTCNN

Una vez se han comentado en los apartados anteriores las herramientas utilizadas por *MTCNN* para realizar su detección, vamos a mencionar el flujo de trabajo que realiza. Inicialmente, como se ha comentado, se obtiene una pirámide de imágenes, de manera, que la de menor tamaño de $12x12x3$ pixeles. A cada imagen se le pasa la red neuronal *P-Net*, que nos aporta las ventanas es las que se encuentran los rostros junto con su confianza. A estas soluciones se le aplica el algoritmo *NMS* con el objetivo de eliminar ventanas redundantes sobre cada imagen. Con todas las soluciones, se vuelve a aplicar el algoritmo NMS y un refinado, eliminando no solo las ventanas sobrantes en una imagen, si no que tambien cohesionando las soluciones iguales dadas en diferentes imágenes escaladas.

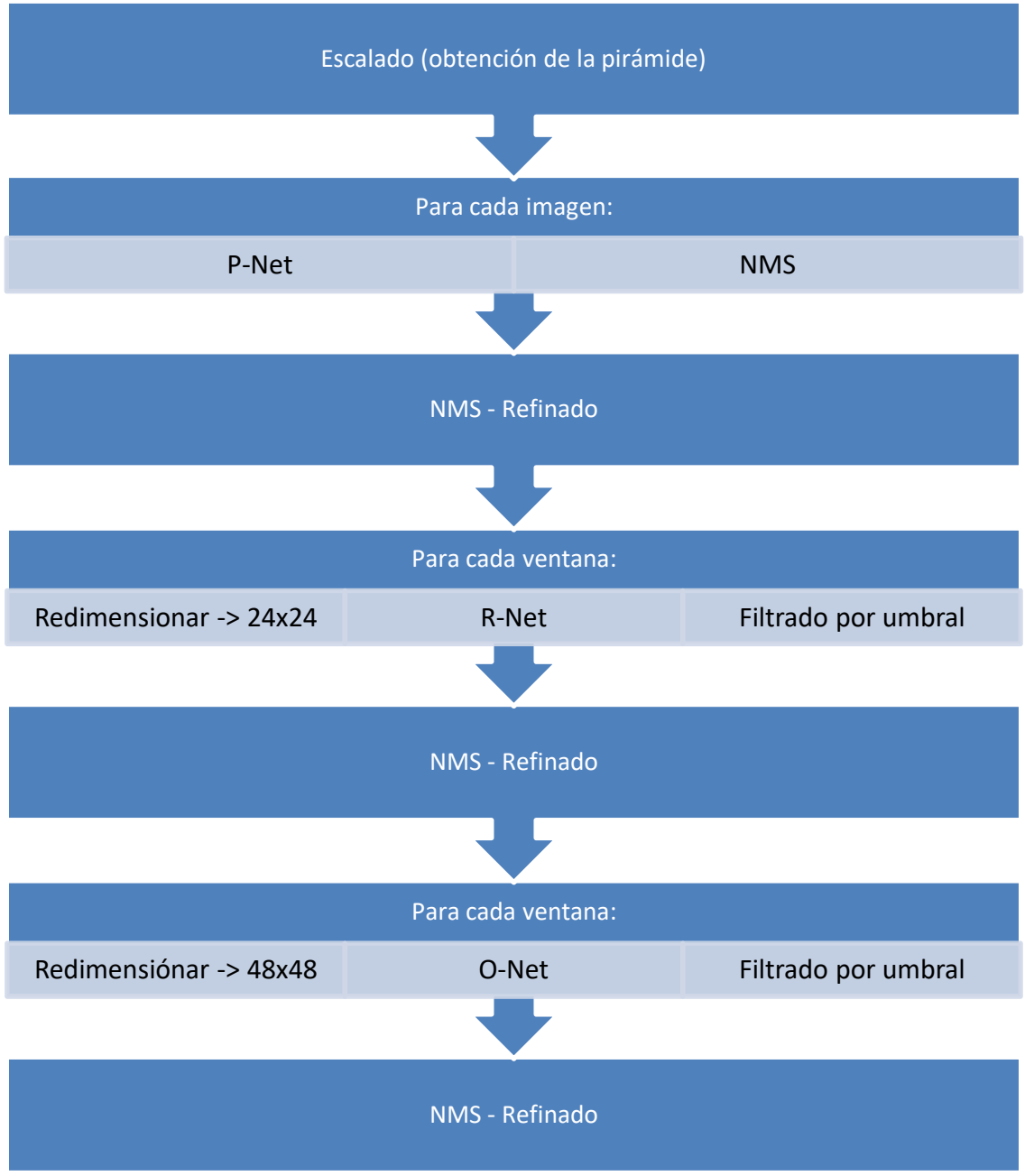

Ilustración 48. MTCNN. Diagrama.

Posteriormente, se modificará el tamaño de cada ventana encontrada, de manera que pase a ser de $24x24$x3 pixeles. A estas, se le aplicará la red *R-Net*, que nos dará las ventanas de una manera más fina junto con su confianza. Solo pasarán a la siguiente etapa aquellas que superen un determinado umbral, en este caso, igual a 0.7. Los resultados obtenidos pasarán por el algoritmo NMS y un refinado. Análogamente, cada solución se vuelve a escalar, consiguiendo un tamaño igual a $48x48x3$ pixeles. Sobre ellas, se aplica la red *O-Net*, cuyas soluciones vuelven a ser filtradas en función de un umbral igual a 0.7. Finalmente, a los resultados finales se les vuelve a aplicar el algoritmo *NMS* y una etapa de refinado.

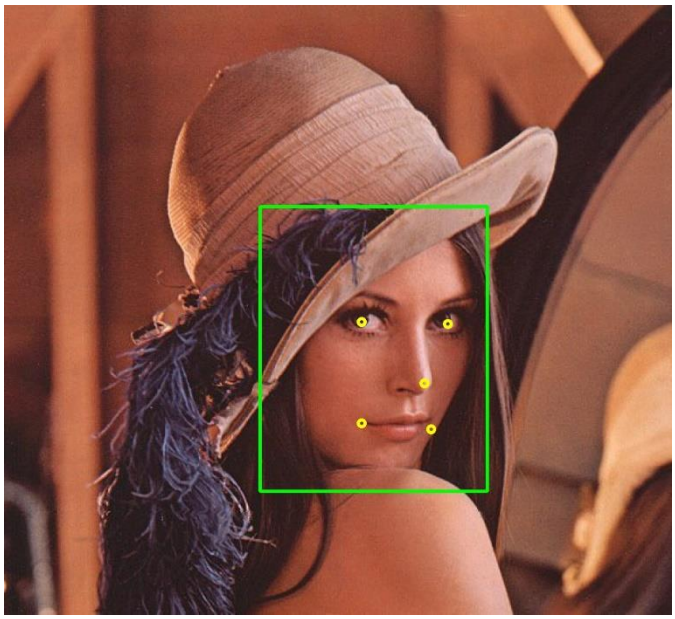

Ilustración 49. Resultado de la aplicación de MTCNN.

Con ello, se obtienen no solo las ventanas en las que se encuentran rostros, sino que tambien 5 puntos característicos de dicha cara. En la Ilustración 49 se puede apreciar el resultado de aplicar dicha red neuronal.

4.3 Comparación entre el detector de rostros MTCNN y el basado en HOG y SVM empleado en DLIB.

Para comparar ambos detectores de rostros se han utilizado dos bases de datos. La primera de ellas es *YouTube Faces Database* [25],que contiene imágenes obtenidas de *frames* de un total de 3425 videos de 1595 personas diferentes. Además, viene complementada con documentos de texto en los que se indica el recuadro contenedor del rostro que se presenta. Un ejemplo de estas son las mostradas en la Ilustración 50. Aunque las imágenes contenidas en este *dataset* no son sencillas, por contener rostros que presentan oclusiones y por la calidad de estas, se ha utilizado otro banco de imágenes cuyas fotografías han sido tomadas en la calle. Este segundo dataset es *GRAZ01-persons* con 460 imágenes, en las cuales no todas ellas contienen rostros. En una imagen puede haber una o más personas, de las cuales no tiene por qué estar presente la cara y, de estarlo, es posible que sea poco visible. Un ejemplo de estas son las mostradas en la Ilustración 51.

Ilustración 50. Ejemplos de las imágenes del dataset "Youtube Faces Database".

Ilustración 51. Ejemplos de imágenes del Dataset "GRAZ01-persons".

Para el primer banco de imágenes (*Youtube Faces Database*), al tenerse el recuadro que delimita los rostros, se ha programado un algoritmo, que mediante la intersección sobre la unión (*IoU*) nos permite establecer si la solución obtenida, de existir, es válida. Con el algoritmo *MTCNN* se ha establecido el umbral de IoU en el 80%. Se han encontrado un total de 446648 rostros positivos de los 621126 totales, es decir, el éxito es del 71.9%. Utilizando el algoritmo de detección de *DLIB*, con un umbral del 80% se obtienen resultados bastante malos, ya que para un total de 27000 rostros solo se detectan 4322 positivos (16%). Con esta tendencia claramente, al realizar todo el proceso el resultado no iba a variar drásticamente, por lo que se ha decidido disminuir la IoU al 50%. En dicho caso, se han encontrado 98325 rostros de los 162000 estudiados, lo que representa un 60,7%. En el mismo experimento, para las mismas 27000 que se examinaron con la *IoU* del 80% se han conseguido 17884 detecciones (66.23%). Claramente, eso indica que los cuadros contenedores dados por *DLIB* varían con respecto a las soluciones dadas por el *dataset* de manera mucho más significativa de lo que lo hace *MTCNN*. En la Ilustración 52 se muestra un gráfico con los resultados obtenidos en los experimentos.

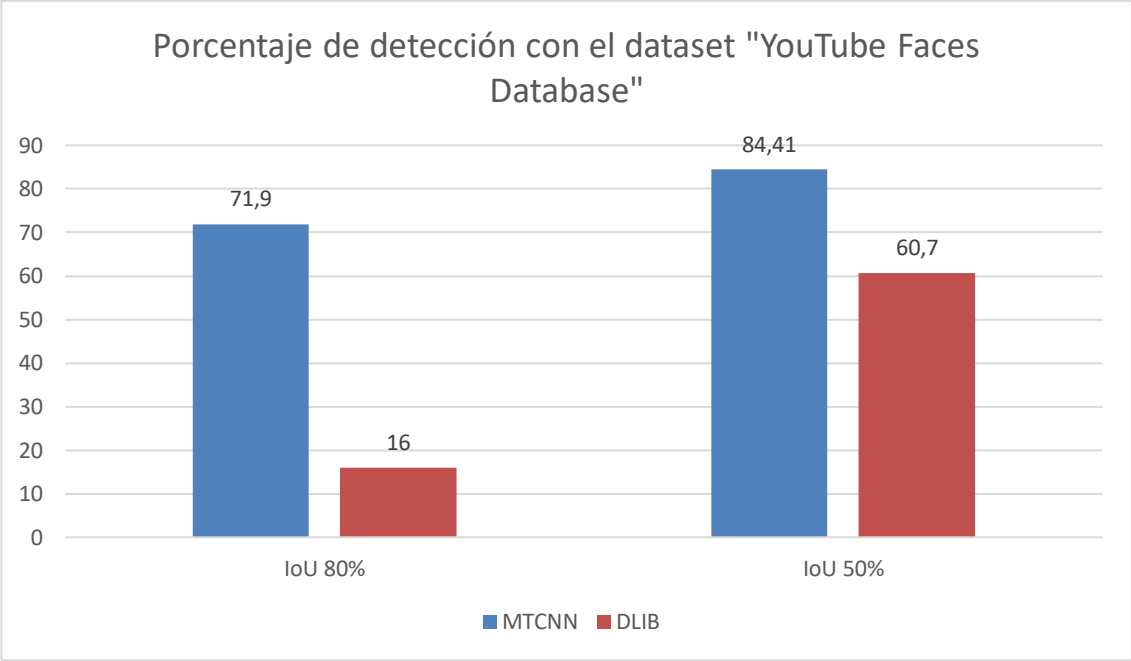

Ilustración 52. Porcentaje de detección con el dataset “YouTube Faces Database”.

En la Ilustración 53 se muestra una de las detecciones realizadas sobre el banco de imágenes *Youtube Faces Database*. En ella, el algoritmo *MTCNN* encuentra adecuadamente el rostro, mientras que DLIB no. Hay que indicar que la calidad de la imagen es reducida, y que el rostro está parcialmente difuso debido a las gafas que posee el hombre. Un resultado similar se aprecia en la Ilustración 54.

Ilustración 53. Resultados obtenidos con MTCNN y DLIB respectivamente usando una imagen del dataset YouTube Faces Database.

Ilustración 54. Resultados obtenidos con MTCNN y DLIB respectivamente usando una imagen del dataset YouTube Faces Database.

En la Ilustración 55 se ha vuelto a presentar el mismo experimento sobre otra imagen. En este caso, *MTCNN* solo encuentra uno de los rostros, ya el otro tiene pocas características visibles. *DLIB* también aporta una solución, que en este caso se corresponde con un falso positivo.

Ilustración 55. Resultados obtenidos con MTCNN y DLIB respectivamente usando una imagen del dataset YouTube Faces Database.

En la Ilustración 56 se puede apreciar como ambos detectores encuentran el rostro. Sin embargo, el recuadro respectivo a la solución otorgada por *MTCNN* representa una mejor solución que el dado por la de *DLIB*. Es uno de los motivos por el cual ha sido necesario aumentar el *IoU* para tener mejores resultados.

Ilustración 56. Resultados obtenidos con MTCNN y DLIB respectivamente usando una imagen del dataset YouTube Faces Database.

En la Ilustración 58 se tienen dos rostros, de los cuales *MTCNN* encuentra ambos, mientras que *DLIB* no lo hace. En la Ilustración 57 se da una situación similar, con la diferencia de que ambos métodos encuentran ambas caras.

Ilustración 57. Resultados obtenidos con MTCNN y DLIB respectivamente usando una imagen del dataset YouTube Faces Database.

Ilustración 58. Resultados obtenidos con MTCNN y DLIB respectivamente usando una imagen del dataset YouTube Faces Database.

Para el segundo banco de imágenes, al no tenerse el cuadrado delimitador del rostro, se ha comprobado manualmente el resultado obtenido con ambos métodos. Este banco de imágenes no presenta solo rostros, sino que personas en las que se aprecia el cuerpo completo, ya sea de espalda o de frente, partes del cuerpo y agrupaciones de individuos en las que no se distinguen las características de la cara. Es por ello por lo que se ha considerado, personalmente, que va a ser un positivo en la imagen, y que no. Por ejemplo, en la cuarta imagen y en la última de la Ilustración 51 no se va a considerar ningún valor certero y tan solo uno (el del chico del centro, para el cual se pueden, parcialmente, distinguir sus características), respectivamente. Los resultados obtenidos son bastante bajos, dando un porcentaje de acierto igual al 46.4% para *MTCNN* y del 16.6% para *DLIB*. En la Ilustración 59 se muestra un gráfico con los resultados comentados. Hay que indicar que el obtener estos datos viene referido a que hay imágenes donde los rostros se encuentran en ventanas de reducidas dimensiones y con grandes niveles de oclusión. En la Ilustración 60 hay un ejemplo de imagen en la que ninguno de los métodos aportó detecciones válidas.

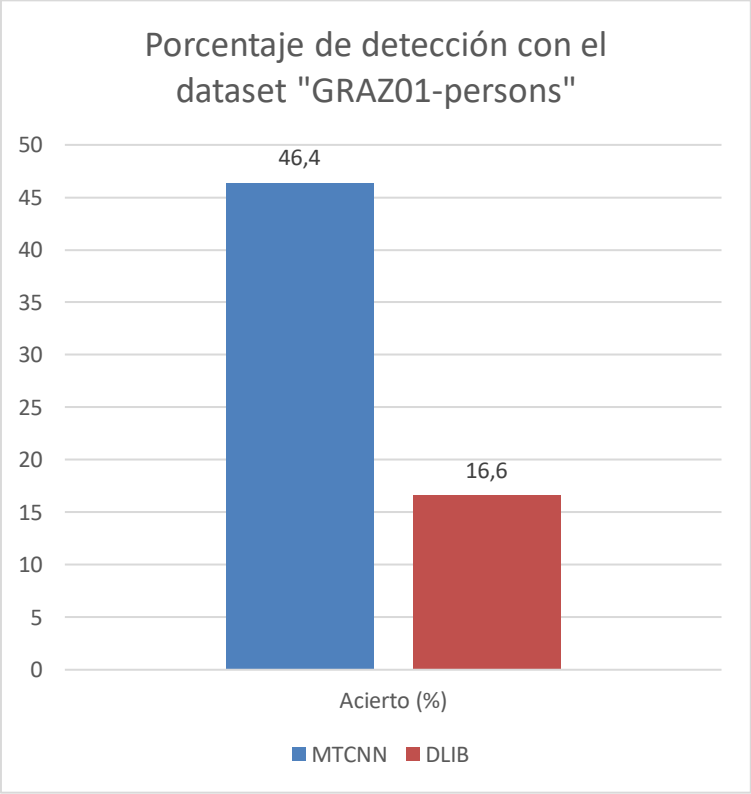

Ilustración 59. Porcentaje de detección con el dataset “GRAZ01-persons”.

En la Ilustración 61 se compara el funcionamiento de ambos métodos con una imagen del banco de imágenes *GRAZ01-persons*. Con *MTCNN* se detectan todos los rostros, mientras que DLIB solo consigue encontrar uno, dando además como solución un cuadrado delimitador bastante erróneo.

Ilustración 60. Imagen sacada del dataset "GRAZ01-persons"

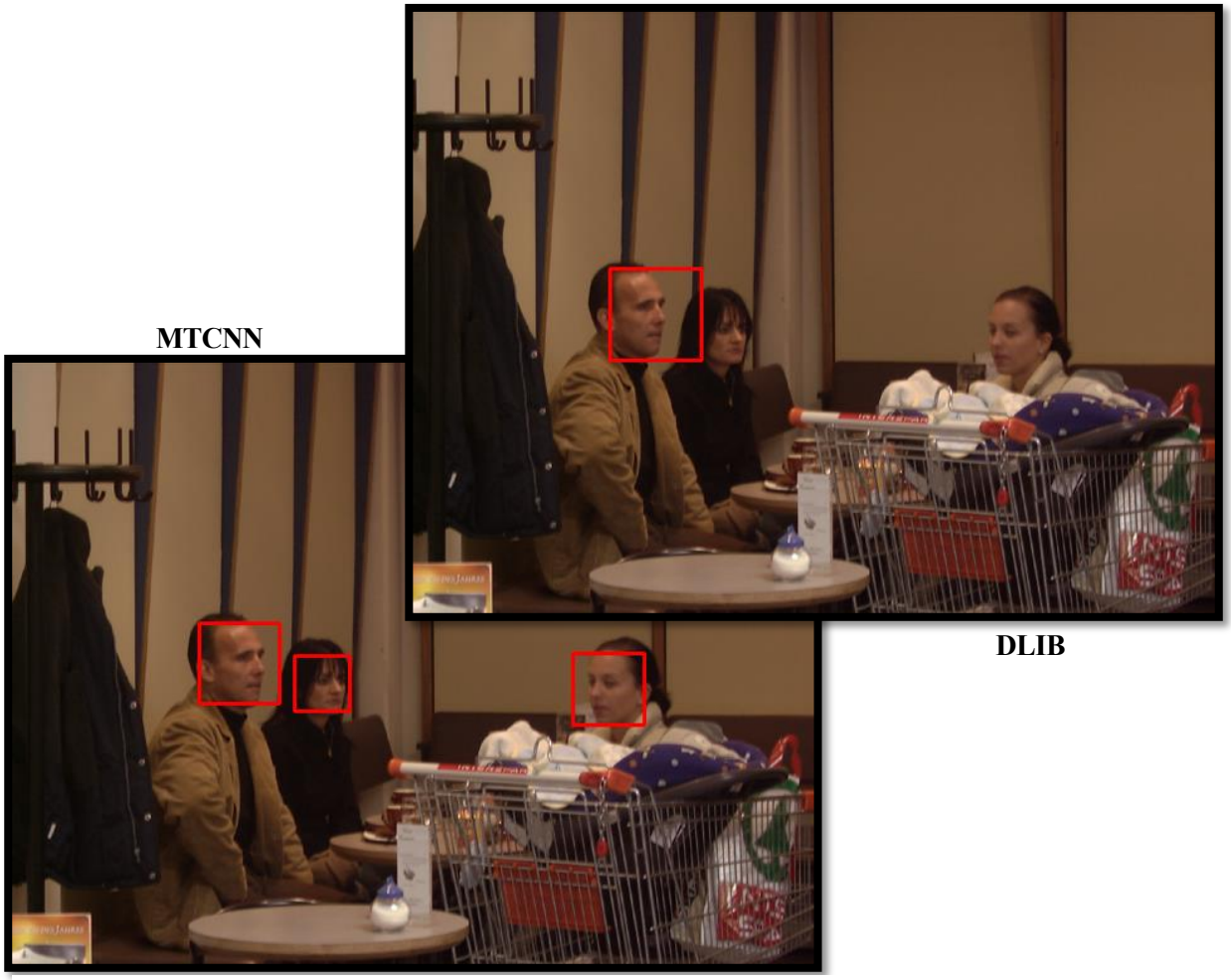

Ilustración 61. Resultados obtenidos con MTCNN y DLIB respectivamente usando una imagen del dataset *GRAZ01-persons*.

En la Ilustración 62 se realiza una comparación similar a la anterior con otra imagen. Es este caso, los rostros más cercanos, que contienen un grado alto de oclusión y que además están en una posición girada, no se encuentran para ninguno de los métodos. En el caso de *MTCNN* se encuentra un rostro, parcialmente girado, pero que contiene un grado bastante bajo de oclusión.

Ilustración 62. Resultados obtenidos con MTCNN y DLIB respectivamente usando una imagen del dataset GRAZ01-persons.

Como conclusión se puede determinar que el método basado en redes neuronales convolucionales (*MTCNN*) ofrece mejores resultados que el ofrecido por *DLIB*, que utiliza el descriptor *HOG* y el clasificador *SVM*. Fundamentalmente se pueden encontrar significativas mejoras ante tamaños del rostro menores, caras no posicionadas frontalmente y oclusiones.

5 RECONOCIMIENTO FACIAL

Preguntarse cuándo los ordenadores podrán pensar es como preguntarse cuándo los submarinos podrán nadar.

Edsger W. Dijkstra

Un sistema de reconocimiento facial es un método automático mediante el cual una máquina identifica a una persona en una imagen. Es decir, su objetivo consiste en, partiendo de una imagen con un rostro "conocido", reconocer este mismo dentro de otra, con una o varias caras que se consideran "desconocidas". Dentro de este, se pueden distinguir los siguientes 3 problemas:

- Verificación: Consiste en determinar si los rostros de dos imágenes diferentes se corresponden entre sí.
- Reconocimiento: Consiste en encontrar a una persona, definida por una imagen de su rostro, comparándola con otras personas, cuyas imágenes se encuentran almacenadas en una base de datos.
- Agrupación (*clustering*): Consiste en congregar imágenes (rostros) cuyas características son similares y, por tanto, se corresponden con la misma persona.

En este caso, la tarea que se debe realizar es de reconocimiento. Para ello se va a utilizar la red neuronal convolucional *FaceNet*, que aporta un conjunto de 128 características que definen un rostro humano.

5.1 FaceNet

FaceNet [15] es una red neuronal convolucional presentada en 2015 por un equipo de *Google* en la que se pretenden resolver los problemas de verificación, reconocimiento y clustering relativos al área del reconocimiento de rostros. Como entrada se recibe una imagen RGB de dimensiones *nxn* y como salida aporta un vector normalizado de características, que definen el rostro dado en la fotografía, de 128 dimensiones.

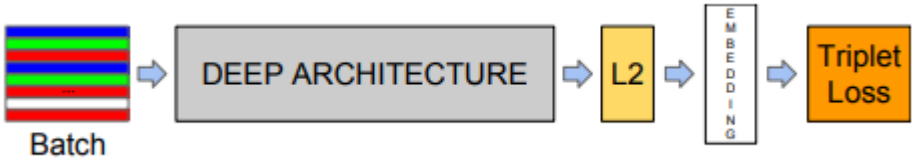

Ilustración 63. Estructura del modelo de FaceNet. Imagen sacada de [15].

La estructura que sigue se muestra en la Ilustración 63. Al inicio, se ve una capa de entrada por lotes, seguida de una red neuronal convolucional, tras las que se encuentra una normalización (*L2*), que nos aporta el resultado buscado. Durante el entrenamiento, al final del modelo se posiciona la triple pérdida (*Triplet Loss*), que es considerada como una de las verdaderas novedades de este método.

Una de las propiedades de FaceNet se encuentra en su alta invarianza ante cambios de iluminación, posición u oclusiones. Además, ante cambios en la gesticulación del individuo obtiene también buenos resultados. La precisión obtenida con el modelo *NM1* para la base de datos *Labeled Faces in the Wild (LFW)* [26] es del 99,63%, mejorando el estado del arte dado por *DeepId2+* en un 30%, mientras que para *YouTube Faces DB* es del 95.12%.

5.1.1 Red neuronal convolucional

Para la arquitectura de la red se propusieron dos modelos, el primero de ellos fue el de *Zeiler&Fergus* [27], denominado *NM1*, que contiene 22 etapas, dentro de las que se incluyen convoluciones, submuestreos, normalizaciones, capas full connected y concatenaciones. Para el ejemplo de la Tabla 1, en el cual se utiliza dicha estructura con convoluciones de dimensión 1x1, para un total de 140M de parámetros se realizan 1.6 billones de operaciones de punto flotante por segundo (FLOPS).

El segundo modelo fue el basado en GoogLeNet style Inception Models [28] que, cuya ventaja con respecto al anterior, contiene una cantidad mucho más reducida de parámetros, y como consecuencia, supone realizar un menor número de operaciones de punto flotante por segundo, lo que posibilita su uso en pequeños dispositivos como *smarthphones*. La red *NM2*, que es la de menor tamaño de este tipo, es descrita en la Tabla 2. De este último hay varios modelos, que van desde el *NM2* hasta el *NM4* y del *NNS1*

al *NNS2*.

layer	size-in	size-out	kernel	param	FLPS
conv1	220×220×3	110×110×64	7×7×3, 2	9K	115M
pool1	110×110×64	55×55×64	3×3×64, 2	0	
rnorm1	55×55×64	55×55×64		0	
conv2a	55×55×64	55×55×64	1×1×64, 1	4K	13M
conv2	55×55×64	55×55×192	3×3×64, 1	111K	335M
rnorm2	55×55×192	55×55×192		0	
pool2	55×55×192	28×28×192	3×3×192, 2	0	
conv3a	28×28×192	28×28×192	1×1×192, 1	37K	29M
conv3	28×28×192	28×28×384	3×3×192, 1	664K	521M
pool3	28×28×384	14×14×384	3×3×384, 2	0	
conv4a	14×14×384	14×14×384	1×1×384, 1	148K	29M
conv4	14×14×384	14×14×256	3×3×384, 1	885K	173M
conv5a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv5	14×14×256	14×14×256	3×3×256, 1	590K	116M
conv6a	14×14×256	14×14×256	1×1×256, 1	66K	13M
conv6	14×14×256	14×14×256	3×3×256, 1	590K	116M
pool4	14×14×256	7×7×256	3×3×256, 2	0	
concat	7×7×256	7×7×256		0	
fc1	7×7×256	1×32×128	maxout p=2	103M	103M
fc2	1×32×128	1×32×128	maxout p=2	34M	34M
fc7128	1×32×128	1×1×128		524K	0.5M
L2	1×1×128	1×1×128		0	
total				140M	1.6B

Tabla 1. Estructura de la red convolucional utilizando filtros de dimensiones $1x1$. Sacada de [15].

type	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj (p)	params	FLOPS
conv1 (7×7×3, 2)	112×112×64	1							9K	119M
max pool + norm	56×56×64	0						m 3×3, 2		
inception (2)	56×56×192	2		64	192				115K	360M
norm + max pool	28×28×192	0						m 3×3, 2		
inception (3a)	28×28×256	2	64	96	128	16	32	m, 32p	164K	128M
inception (3b)	28×28×320	2	64	96	128	32	64	L_2, 64p	228K	179M
inception (3c)	14×14×640	2	0	128	256,2	32	64,2	m 3×3,2	398K	108M
inception (4a)	14×14×640	2	256	96	192	32	64	L_2, 128p	545K	107M
inception (4b)	14×14×640	2	224	112	224	32	64	L_2, 128p	595K	117M
inception (4c)	14×14×640	2	192	128	256	32	64	L_2, 128p	654K	128M
inception (4d)	14×14×640	2	160	144	288	32	64	L_2, 128p	722K	142M
inception (4e)	7×7×1024	2	0	160	256,2	64	128,2	m 3×3,2	717K	56M
inception (5a)	7×7×1024	2	384	192	384	48	128	L_2, 128p	1.6M	78M
inception (5b)	7×7×1024	2	384	192	384	48	128	m, 128p	1.6M	78M
avg pool	1×1×1024	0								
fully conn	1×1×128	1							131K	0.1M
L2 normalization	1×1×128	0								
total									7.5M	1.6B

Tabla 2. Estructura de NM2 Inception, para FaceNet. Tabla sacada de [15].

5.1.2 Triplet Loss

Uno de los conceptos más relevantes de *FaceNet* es el de la triple pérdida (*Triple Loss*) según la cual es entrenada la red neuronal convolucional. Una imagen, con esta red, se proyecta en un espacio euclideo d-dimensional y restringido a una hiperesfera. Lo que se pretende es que una determinada imagen x_i^a, denominada *ancla* (*anchor*), se encuentre a una distancia menor de imagen positiva x_i^p, que de otra negativa x_i^n (Ilustración 64). El objetivo de esto es el de aumentar la capacidad discriminativa que tendrá la *CNN*. En la Ecuación 18 se muestra la ecuación que modela este proceso, en ella, α es un umbral entre los pares positivos y negativos y T el conjunto de todas las posibles tripletas en el conjunto de entrenamiento. La pérdida se minimiza cuando se cumple la Ecuación 19.

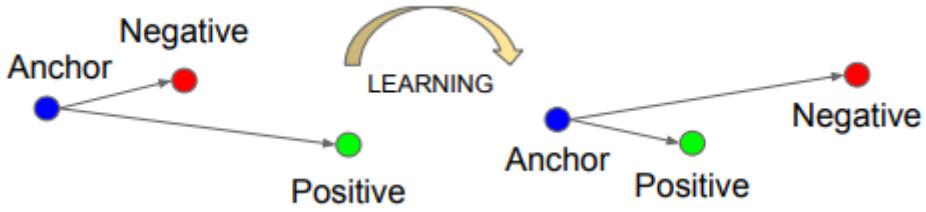

Ilustración 64.Triplet Loss. Esquema de funcionamiento. Imagen sacada de [15].

$$\left\| f\left(x_i^a - f\left(x_i^p\right)\right)\right\|_2^2 + \alpha < \left\| f\left(x_i^a - f\left(x_i^n\right)\right)\right\|_2^2, \forall \left(f\left(x_i^a\right), f\left(x_i^p\right), f\left(x_i^n\right)\right) \in T$$

Ecuación 18. Tripletas

$$L = \sum_i^N \left[\left\| f\left(x_i^a - f\left(x_i^p\right)\right)\right\|_2^2 - \left\| f\left(x_i^a - f\left(x_i^n\right)\right)\right\|_2^2 + \alpha \right]_+$$

Ecuación 19. Minimización de la pérdida.

El uso de todo el conjunto completo de tripletas provoca que la convergencia de la red sea muy lenta, muchas de ellas no aportando positivamente en el entrenamiento. Por esta razón es importante elegir bien qué tripletas se utilizan. Para ello, el enfoque utilizado fue el de escoger para cada imagen aquella otra imagen positiva con la cual la distancia fuese máxima y una negativa con respecto a la cual la distancia fuese mínima. El problema que esto conlleva es la dificultad de calcular dichas distancias en todo el conjunto de entrenamiento, además del efecto negativo ocasionado por rostros mal etiquetados o representados que entrarían a formar parte de las tripletas. Como solución se generan pequeños lotes con muestras positivas y negativas, calculando las distancias entre ese reducido número de imágenes.

5.2 Prueba de FaceNet

Con el objetivo de probar el funcionamiento del algoritmo de reidentificación de rostros se han realizado varios programas sencillos en *Python*. Para ello, se ha utilizado el banco de imágenes *YouTube Faces Database*. Este está compuesto por imágenes sacadas de 3425 videos de dicha plataforma de 1595 personas. Se siguieron distintos criterios para realizar las comparaciones.

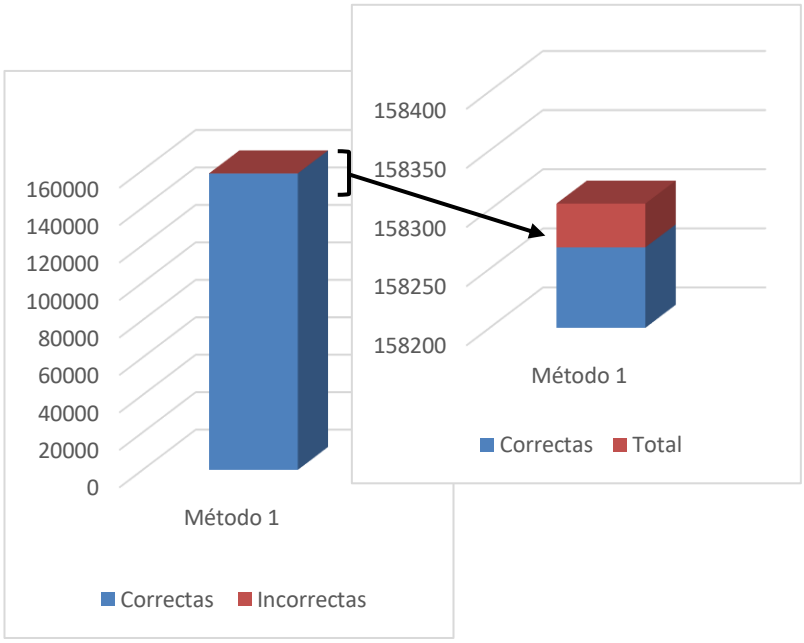

Ilustración 65. Prueba de FaceNet según el método 1.

En primer lugar, para un total de 158305 imágenes sacadas de este *dataset* se buscó aquella en la que aparecía la persona que más se la asemejaba. Si ambas se correspondían con la misma persona se consideraba como un acierto. Como es de esperar, los resultados obtenidos con este enfoque fueron muy buenos. Esto se debe a que, al estar las imágenes sacadas de *frames* de un mismo video, la diferencia entre el rostro de imágenes consecutivas es pequeña, generando un acierto. Para sacar estos datos solo se tuvieron en cuenta aquellas identificaciones que ya había generado previamente un acierto en lo relativo al problema de detección de rostros. De esta manera, aquellos casos en los que este problema fallase no han sido considerados como error. El número de rostros identificados correctamente, como puede verse en la Ilustración 65, es igual a 158268. Por tanto, el porcentaje de detecciones correctas es igual al 99.9766%.

Posteriormente, se decidió utilizar tan solo a aquellas personas que tenían imágenes sacadas de varios videos, de manera que un conjunto pudiese utilizarse solo como modelo. De esta forma, las imágenes con las que se buscaban correspondencias no se comparaban con una que ha sido tomada del mismo video y, por tanto, que se le parece considerablemente. Los resultados en este caso cayeron, siendo la tasa de acierto considerablemente más baja. Como puede verse en la Ilustración 66, el número de rostros identificados correctamente es igual a 21092, para un total de 82675. Por tanto, el porcentaje de detecciones correctas es igual al 25%. Al ver el efecto que tenía sobre las imágenes el proceso de identificación del individuo se pudo observar cómo, debido a distintos factores, como pueden ser oclusiones, cambios en las expresiones, mala iluminación o desenfoques, la tasa de acierto disminuía considerablemente, siendo este decremento mayor cuando el modelo escogido era el que sufría alguno de ellos. Si, además, el número de imágenes tomadas de dicho video era elevado, el impacto sobre el porcentaje de acierto también era mayor. En la Ilustración 67 puede verse una de las imágenes para las cuales se trata de realizar la tarea de reidentificación. Puede observarse como es una imagen compleja, incluso para el ojo humano, ya que presenta un rostro parcialmente oculto y poco nítido.

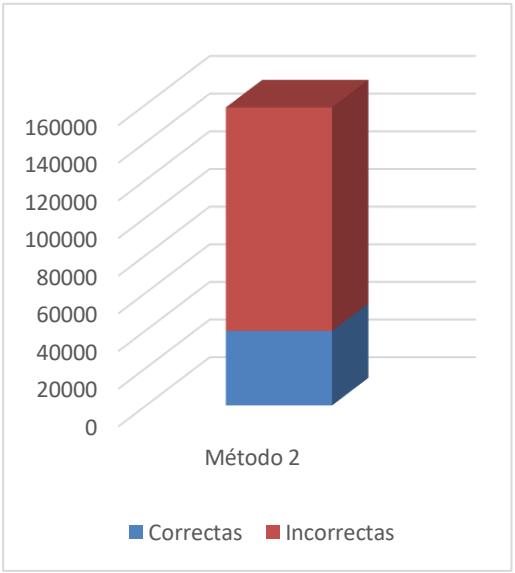

Ilustración 66. Prueba de FaceNet según el método 2.

De una manera más controlada, se han realizado una tercera prueba, escogiendo para ella la imagen modelo que define al individuo, es decir, entre las videos asociados a cada una de las personas, se ha escogido aquel en el que el rostro que aparece está en una posición más frontal y con menos oclusiones. Para esta prueba se han utilizado un total de 100 individuos, con un total de 21320 imágenes entre todos. Además, esta vez se ha evaluado el problema completo, considerando como erróneo aquel caso en el que ni siquiera se detecta un rostro. El programa que se ha utilizado es uno similar al descrito en el apartado 6.3, cuya única diferencia es que realiza pruebas de forma masiva, es decir, lee las imágenes que se encuentran en una carpeta, y sobre ellas realiza el algoritmo indicado en dicho punto. Para una mayor claridad, esta prueba será explicada en el apartado . Se puede adelantar, que el porcentaje de acierto es del 73.6%.

Ilustración 67. Imagen obtenida del dataset.

A continuación, se van a mostrar una serie de imágenes para ejemplificar los efectos encontrados.

Ilustración 68. Detección correcta para todos los tipos de imágenes de un individuo (modelo → arriba-izquierda).

Ilustración 69. Detección de un individuo (izquierda) utilizando como imagen modelo (derecha) una cuyo rostro se encuentra parcialmente oculto.

Ilustración 70. Detección correcta en una imagen (izquierda) con el rostro parcialmente oculto a partir de la imagen con la cara modelo (derecha) situada frontalmente.

En la Ilustración 68, se puede observar como todas las imágenes de pruebas, ilustradas mediante la representación de una por cada video, aportan la identificación correcta.

Finalmente, en la Ilustración 69 se puede observar cómo se tiene un modelo (derecha) cuyo rostro se encuentra parcialmente oculto, al intentar detectar la persona que se encuentra en la imagen de prueba (izquierda) el programa nos indica que esta es desconocida. En la Ilustración 70, se parte de un modelo frontal (derecha), pero la cara con la cual se realiza la comparación (izquierda) está colocada lateralmente. A pesar de esto, se consigue detectar la identidad del individuo. Hay que comentar que también nos encontramos con una persona, sin introducir en la base de datos, identificada, certeramente, como desconocida. En la Ilustración 71 se pueden ver dos imágenes, una perteneciente al modelo (izquierda) y otra de prueba que, según indica el *dataset* se corresponden con el mismo individuo. Hay que indicar que las diferencias son notables y que, comprensiblemente, el algoritmo da lugar para todas las imágenes detecciones erróneas, asignándole la identidad del individuo de la Ilustración 68.

Ilustración 71. Detección errónea (modelo arriba-izquierda).

6 DETECTORES IMPLEMENTADOS

Crear un ser artificial ha sido el sueño del hombre desde que nació la ciencia

Película (A.I Inteligencia artificial)

En este apartado se va a comentar detalladamente la implementación de los archivos *detector_mtcnn.cpp*, *detector_mtcnn.h*, *detector_mtcnn_facenet.cpp*, *detector_mtcnn_facenet.h* y *main.cpp* (del subdirectorio *detector/testFacenet*). Para ello, también se tendrá que mencionar cual es la filosofía del resto del espacio de trabajo. En lo que atañe al proyecto hay que tener en cuenta archivos incluidos en la carpeta *Data*, en concreto *Data::CFace*, ya que nos determinan el formato de la salida aportado por los detectores. Además, se comentará el funcionamiento de *detector_manager*, ya que hace de puente entre los archivos *main* y los propios detectores. También se tendrán que considerar los archivos *.Json*, almacenados en *detector/cfg*, que contienen la configuración de los detectores. En esta misma ubicación, aunque podría realizarse en otra indicándolo correctamente en los ficheros que se acaban de comentar, se han introducido previamente cada uno de los archivos que contienen los pesos o la configuración de las redes neuronales. Finalmente, se tendrá que comentar la existencia de un fichero principal, encargado de realizar detecciones, dentro de la carpeta *detector/test*. Este es similar al creado para la identificación de individuos (apartado 6.3). La principal diferencia práctica es que, cuando llama al detector vinculado con *FaceNet* tan solo se obtiene la estructura de datos relacionada con los descriptores asociados a los individuos que se encuentran en la imagen, y no su identificación con respecto a todos aquellos almacenados en la base de datos. Para conseguir esto se utiliza el compilado *detector_facenet_test*, obtenido del archivo ubicado en *detector/testFacenet*.

6.1 Detector MTCNN

Para la elaboración de este detector se ha creado una clase, llamada *MTCNN*, según la estructura del resto de clases pertenecientes a detectores proporcionadas por *4i*. Esta está compuesta por un método encargado de realizar la inicialización de la red, que tiene como entrada un archivo *.Json*, descrito más adelante y la dirección absoluta a la carpeta *detector*. La función de este es la de obtener todas las rutas a los ficheros que contienen los pesos de las redes neuronales *PNet, RNet* y *Onet* descritas en el apartado 4.2. Además, también contiene otro método, *detect*, que se encarga de cargar la imagen en la cual se deben realizar las detecciones. Posteriormente se llama a la clase *mtcnn* y se invoca el método encargado de la detección de rostros. Para cada *Bounding Box* encontrado se modifica el valor de los pixeles que lo definen, pues la imagen ha cambiado de tamaño, y se almacena, junto con la probabilidad obtenida de que sea un rostro, en un puntero de tipo *data::CFace*. El conjunto de estos punteros es devuelto por dicho método.

Para probar su funcionamiento se utiliza el compilado *detector_test*, aportado por *4i*. A continuación, se indica la instrucción que debe ser incluida por terminal para obtener la posición de los rostros dentro de una imagen.

```
./detector_test --image <ruta de la imagen> --config  <ruta de la carpeta contenedora del archivo .Json>/detector_mtcnn.json'  --classes face
```

6.1.1 JSON asociado a MTCNN

El archivo *detector_mtcnn.json* se utiliza para indicar el tipo de detección que se desea, "mtcnn" en dicho caso, la clase que se está buscando, "face", y las rutas a los pesos de las redes neuronales *PNet, RNet y ONet*. Este puede verse en la Ilustración 72.

```
{
    "overlapping_threshold" : 0.75,
    "detectors":
[
  {
    "type" : "mtcnn",
    "classes" : ["face"],
    "detector_config" :
    {
        "pNetPath" : "../../detector/cfg/Pnet.txt",
        "rNetPath" : "../../detector/cfg/Rnet.txt",
        "oNetPath" : "../../detector/cfg/Onet.txt"
    }
  }
    ]
}
```

Ilustración 72. JSON del detector MTCNN.

6.2 Detector MTCNN junto con FaceNet.

Este archivo mantiene una estructura similar a la descrita en el apartado 6.1. Como ya ocurría, se tiene un método encargado de la inicialización del detector en el que se cargan las rutas correspondientes a archivos externos. Además de las comentadas anteriormente, será necesario obtener, y por tanto recibir, las vinculadas con el modelo de la red neuronal *FaceNet*. En este caso, se crea un Tensor de tipo flotante y tamaño conocido (160x160x3) y se lee la definición del modelo aportado, además de crear una sesión para poder utilizarlo. El método *detect*, recoge la imagen de la cual se quieren obtener los rostros junto con sus descriptores, instancia la red neuronal *MTCNN*, y detecta con ella las caras sobre dicha figura. Para cada uno de los *Bounding Box* encontrados se modifica su tamaño a uno predefinido, e igual al del Tensor anterior (160x160). Posteriormente se pasa a modificar los datos de entrada a la red neuronal, como se hizo con *MTCNN* en el apartado 4.2. Para ello se copia todo en un vector, del que se calcula la media y la desviación estándar. Con estos valores, a los valores de los pixeles de la imagen original se le realiza una sustracción de la media, dividiéndolos además por su desviación estándar. Se consigue así que siga una distribución normal. Los datos obtenidos son pasados al Tensor inicial. Para cada uno de estos tensores se ejecuta la red, con dos capas de entradas, una de las cuales tiene como entrada el tensor calculado, y otra con un nuevo tensor. La salida, correspondiente con el descriptor de cada rostro, se almacena en un vector de flotantes. Al igual que ocurría anteriormente se calcula el *bounding*

box real sobre la imagen de entrada y se crea un elemento, de tipo *data::CFace,* al que se le asocia el rectángulo real al que pertenece el rostro, un identificador, y el vector de características obtenido.

Con el objetivo de probar su funcionamiento se utiliza el compilado *detector_test*, al igual que se hizo en el apartado 6.1. A continuación, se indica la instrucción que debe ser introducida por terminal para obtener el descriptor de cada uno de los rostros que se encuentran dentro de la imagen.

```
./detector_test --image <ruta de la imagen> --config  <ruta de la carpeta contenedora del archivo .Json>/detector_mtcnn.json'  --classes face
```

6.2.1 Json asociado al detector MTCNN junto con FaceNet

El archivo *detector_facenet.json* se utiliza para indicar el tipo de detección que se desea, "facenet" en dicho caso, la clase que se está buscando, "face", las rutas a los pesos de las redes neuronales *PNet, RNet y ONet*¸la ruta al archivo de configuración de la red FaceNet, el tamaño predefinido del tensor de entrada, así como la definición de sus capas. Este puede verse en la Ilustración 73.

```
{
    "overlapping_threshold" : 0.75,
    "detectors":
[
  {
    "type" : "facenet",
    "classes" : ["face"],
    "detector_config" :
    {
        "model" : "../../detector/cfg/facenet.pb",
        "modelsImagePath" : "../../img/",
        "threshold" : 1.5,
        "heigh" : 160,
        "width" : 160,
        "depth" : 3,
        "input_layer_1" : "input:0",
        "input_layer_2" : "phase_train:0",
        "output_layer" : "embeddings:0",
        "pNetPath" : "../../detector/cfg/Pnet.txt",
        "rNetPath" : "../../detector/cfg/Rnet.txt",
        "oNetPath" : "../../detector/cfg/Onet.txt"
    }
  }
    ]
}
```

Ilustración 73.JSon del detector MTCNN junto con FaceNet.

6.3 Fichero principal para las pruebas con MTCNN y FaceNet

Este archivo se encuentra pensado para realizar pruebas con ambas redes neuronales. Para su correcto funcionamiento es necesario que reciba como parámetros de entrada el fichero JSon al que está vinculado, y que en este caso se corresponde con el indicado en el apartado 6.2; la imagen sobre la cual se quiere realizar la detección de los individuos, el archivo JSon en el que se han almacenado los descriptores de las imágenes modelo y el valor del umbral que delimita si la distancia es suficiente para hacer una asociación.

Una vez se comienza con la ejecución se instancia la clase *CDetectorManager* y se realiza su inicialización, pasándole como argumento de entrada el archivo JSon de configuración. El funcionamiento de este método, superficialmente, se basa en leer el fichero *.json* para así conocer el tipo de detector que se está utilizando. En función de esto crea un puntero a la clase asociada al detector que se encarga de realizar las operaciones pertinentes, en nuestro caso, será de tipo *CDetectorMtcnnFacenet*. Posteriormente, se llama al método de la inicialización de dicho descriptor. Si esta ha sido realizada con éxito se almacenan todas las clases en un vector privado perteneciente a dicha clase junto con los punteros creados.

Volviendo al fichero *main.cpp*, el siguiente paso que se realiza es la carga de la imagen cuya ruta ha sido pasada por terminal. Tras esto, se invoca al método *detect* de *detectorManager*. Este recibe como entrada la imagen sobre la cual realizar las detecciones y una lista con las clases que se están buscando en estas. Su función se enfoca en, para cada puntero asociado a detectores (se tienen todos pues, como se ha comentado, se han almacenado junto con sus clases) y pertenecientes a una de las clases indicadas como entrada al método, llamar al método detect asociado al mismo. Este nos aportará un puntero a un resultado, de tipo *data::AObject* que, en nuestro caso tendrá los datos de uno de tipo *data::CFace*. El conjunto de todos los resultados será devuelto a la función principal.

Gracias a esta salida, y a los descriptores almacenados en el *.json* se puede calcular la distancia euclidea, asignando a cada detección el modelo con el cual la distancia es mínima. Además, se ha incluido un valor umbral, pudiendo ser el individuo desconocido.

Con el objetivo de obtener un resultado más visual en este proyecto se han añadido sobre la imagen de entrada los rectángulos contenedores de rostros, junto con el Id correspondiente a cada vinculación.

Para probar su funcionamiento, en este caso, se utiliza el compilado *detector_facenet_test*. A continuación, se indica la instrucción que debe ser introducida por terminal para obtener la identidad de cada uno de los individuos que se encuentran en la imagen.

```
./detector_facenet_test --image <ruta a la imagen sobre la que se desea identificar rostros>
--config <ruta de la carpeta contenedora de los archivos .Json>/detector_facenet.json
--descriptors <ruta de la carpeta contenedora de los archivos .Json>/facesDescriptions.json
--thr 1.1
```

6.3.1 Pruebas realizadas con distintas imágenes

Como se comentó en el apartado 5.2 se ha realizado una prueba utilizando los archivos comentados en esta sección. Para ello, se han calculado los descriptores de 100 imágenes modelo con el programa *detector_test*. Estos han sido almacenados en el fichero *.Json* que se comentó en 6.3. Con un programa similar a *detector_facenet_test*, cuya única diferencia se encuentra en que realiza un proceso masivo de identificaciones, se ha comprobado si las asignaciones entre los rostros contenidos en las imágenes de prueba y los modelos son correctas. Los resultados obtenidos pueden verse en la Tabla 3, donde se indica en la primera columna el individuo al que pertenecen, en la segunda la distribución de carpetas que contenían el conjunto de imágenes, la tercera el número de identificaciones correctas y la cuarta el número de veces en el que aparece el individuo. Finalmente, en la última fila se realiza el sumatorio de los dos últimos datos. Como se puede observar, el resultado que se ha obtenido es de 15692 rostros identificados con respecto a 21320 totales. Es decir, se ha obtenido un porcentaje de identificaciones correctas del 73.6% para este conjunto de datos.

Número del individuo	Carpeta	Identificaciones correctas	Número de veces en las que aparece el individuo
1	1	49	49
	2	17	56
2	1	0	174
3	1	0	83
4	1	237	237
5	1	12	285
	1	53	141
6	1	0	195
7	1	58	58
	2	63	153
8	1	39	52
	2	26	104
9	1	0	58
10	1	17	48
	2	57	62
11	1	58	58
	2	71	71
	3	55	55
12	1	71	71
	2	39	70
	3	52	52
13	1	51	51
	2	8	80
	3	59	59

14	**1**	**0**	**120**
	2	**0**	**129**
	3	**0**	**144**
15	**1**	**73**	**73**
	2	**83**	**95**
	3	**216**	**233**
16	**1**	**41**	**76**
	2	**85**	**87**
	3	**52**	**64**
17	**1**	**65**	**74**
	2	**60**	**63**
	3	**59**	**70**
18	**1**	**58**	**79**
19	**1**	**54**	**54**
20	**1**	**33**	**53**
21	**1**	**110**	**110**
22	**1**	**55**	**56**
	2	**424**	**432**
23	**1**	**3**	**119**
24	**1**	**31**	**87**
25	**1**	**0**	**101**
	2	**69**	**69**
26	**1**	**39**	**73**
	2	**29**	**75**
27	**1**	**48**	**50**
28	**1**	**338**	**340**
	2	**562**	**629**
29	**1**	**445**	**445**

	2	**500**	**545**
30	**1**	**685**	**685**
31	**1**	**74**	**74**
32	**1**	**13**	**67**
	2	**0**	**125**
33	**1**	**100**	**100**
	2	**61**	**61**
34	**1**	**154**	**154**
	2	**62**	**119**
35	**1**	**32**	**50**
	2	**50**	**51**
	3	**90**	**144**
36	**1**	**97**	**141**
	2	**0**	**56**
	3	**7**	**279**
37	**1**	**82**	**97**
	2	**104**	**104**
	3	**43**	**54**
38	**1**	**108**	**125**
	2	**0**	**101**
39	**1**	**0**	**55**
	2	**0**	**93**
40	**1**	**1**	**86**
	2	**111**	**115**
	3	**47**	**63**
41	**1**	**169**	**168**
42	**1**	**54**	**54**

	2	96	96
	3	50	50
43	1	71	81
	2	0	61
44	1	49	49
45	1	175	226
46	1	103	130
	2	2	94
	3	315	490
47	1	65	65
	2	46	86
48	1	64	64
	2	756	761
	3	91	91
	4	69	69
49	1	93	93
	2	61	61
	3	277	277
50	1	39	68
	2	61	113
51	1	294	294
	2	68	69
	3	57	57
52	1	102	102
	2	190	199
53	1	73	73
54	1	19	55
55	1	112	112

	2	31	55
	3	12	48
56	1	5	66
	2	244	364
57	1	170	198
58	1	129	136
	2	39	53
59	1	202	202
60	1	57	68
	2	49	68
	3	53	109
61	1	0	48
62	1	51	82
63	1	48	53
64	1	69	69
65	1	12	100
	2	165	165
66	1	46	49
	2	93	93
67	1	69	69
	2	129	129
68	1	169	183
	2	116	116
	3	114	117
69	1	108	118
	2	165	165
70	1	62	62

	2	71	76
71	1	0	98
	2	62	69
72	1	114	114
	2	72	73
73	1	93	93
74	1	153	153
	2	100	100
75	1	29	103
76	1	70	72
	2	0	78
77	1	68	70
	2	56	61
78	1	73	73
	2	52	52
79	1	76	76
80	1	90	94
	2	16	81
81	1	53	53
82	1	57	57
	2	72	73
83	1	6	49
	2	0	149
	3	0	127
84	1	61	70
85	1	69	69
	2	40	49
86	1	87	87

	2	**0**	**57**
87	**1**	**189**	**189**
88	**1**	**48**	**48**
89	**1**	**25**	**61**
	2	**114**	**118**
	3	**20**	**73**
90	**1**	**53**	**57**
	2	**60**	**134**
91	**1**	**62**	**62**
	2	**164**	**173**
	3	**117**	**117**
92	**1**	**51**	**53**
93	**1**	**67**	**67**
94	**1**	**37**	**49**
95	**1**	**66**	**66**
96	**1**	**90**	**111**
	2	**3**	**105**
97	**1**	**47**	**49**
98	**1**	**161**	**200**
99	**1**	**63**	**101**
100	**1**	**42**	**52**
Suma		**15692**	**21320**

Tabla 3. Resultados obtenidos con el identificador de rostros para un total de 100 individuos.

Las imágenes que se van a indicar a continuación se corresponden con ejemplos de la prueba que acaba de comentarse.

Ilustración 74. Fallo en las identificaciones o detecciones.

En la Ilustración 74 se pueden observar dos imágenes con dos individuos. En la primera, la mujer tiene asociada una identidad que no le pertenece. Como se verá más adelante, esto podría seguramente solucionarse modificando el valor umbral. En la segunda, solo se detecta un rostro, por lo que en este caso el error se debe a la red neuronal *MTCNN*. Fallos en el problema de detección de caras es otro motivo por el cual la reidentificación de individuos puede dar lugar a malos resultados.

En la Ilustración 75 se muestra un conjunto de imágenes en las cuales todas las identificaciones se han realizado correctamente, incluidas las de los individuos desconocidos.

Ilustración 75. Conjunto de imágenes en las cuales todas las identificaciones se realizan correctamente.

En la Ilustración 76 hay un conjunto de imágenes representativo de una serie de cosas. Por un lado, la primera imagen se corresponde con el modelo del individuo. En la segunda, se realiza una correcta distinción de mismo, asignando a la otra persona la identidad de desconocida. En la tercera y en la cuarta se muestra la variación del valor umbral de FaceNet. Por un lado, en la tercera se acierta al considerar como desconocida a la enfermera, pues su rostro no se encuentra en el archivo *.Json*, sin embargo, no se detecta a *H. Ford*, quien sí que tiene el descriptor incluido en el mismo. Por otro, en la cuarta se ha aumentado el umbral, de 1.1 a 1.5, llegándose a asignar una identidad errónea a las personas. En la quinta y en la sexta se detectan los rostros adecuadamente. Estas se han incluido pues pertenecen al mismo video, y mientras que la sexta es una imagen compleja la quinta no lo es. El motivo por el que se ponen es porque no siempre las imágenes son similares dentro de la misma carpeta donde se han agrupado los *frames* de una grabación.

Similarmente, en la Ilustración 77 se puede ver el cambio en los resultados a modificar el valor umbral, siendo en el primero de los casos igual a 1.2 y en el segundo a 0.9 (valor muy restrictivo).

Ilustración 76. Detección para un individuo incluyendo las variaciones del umbral.

Ilustración 77. Modificación de resultados variando el umbral de FaceNet.

7 CONCLUSIONES Y TRABAJO FUTURO

Sólo podemos ver poco del futuro, pero lo suficiente para darnos cuenta de que hay mucho que hacer.

Alan Turing

Como se ha ido comentando a través de los puntos de este documento, las redes neuronales, son un método de gran importancia y utilizada a la hora de resolver distintos problemas, como aquellos vinculados con la visión por computador. En este caso, nos hemos centrado en aquellos ocasionados por la detección e identificación de rostros.

Para el primero de ellos, en la actualidad existen multitud de soluciones, muchas de las cuales ofrecen resultados realmente buenos. Un ejemplo es la red neuronal *MTCNN*, la cual se trata en este proyecto. Los resultados que se han obtenido, desde el punto de vista del rendimiento, difieren bastante en función del dataset que sea usado, pasando de un 84.4% cuando se utiliza *YouTube Face Database*, a un 46.4%, cuando se utiliza *GRAZ-01*. De esto se puede inferir que aspectos como la calidad de la imagen, el tamaño del rostro dentro de la misma, posibles oclusiones y una posición no frontal pueden afectar considerablemente. Además, de esta red neuronal, se ha utilizado un detector perteneciente a la librería *DLIB* que utiliza como características las dadas por *HOG* y como clasificador *SVM*. El efecto encontrado al probarlo con ambos datasets era similar al visto con *MTCNN*, con la diferencia de que el porcentaje de clasificaciones correctas era sustancialmente peor (60.7%-16.6%).

Para el segundo problema las soluciones encontradas son inferiores, aunque es un campo que en la actualidad está creciendo mucho. En este proyecto se ha utilizado la red neuronal *FaceNet*, que nos da un vector de 128 características con el que poder, mediante la distancia euclidea, comparar rostros. La solución ofrecida por esta tiene acoplados los problemas anteriormente mencionados para *MTCNN*, añadiendo además aspectos relativos a cambios de expresión, que pueden llegar a mover las características, maquillaje… Además, aunque para conjuntos de datos pequeños los resultados que se adquieren son buenos, a medida que el número de clases (rostros modelo) aumenta estos empiezan a empeorar, llegándose a identificar erróneamente a los individuos en un elevado número de casos. Como se ha comentado en el apartado 5.2, existe una gran dependencia entre el resultado obtenido y la imagen modelo escogida, siendo peor cuando esta no representa adecuadamente al individuo. Este es uno de los motivos que explica que haya tanta diferencia entre distintas pruebas realizadas

con *FaceNet.* Se tiene también que comentar que la dependencia que tiene con la correcta identificación y alineación de los rostros en la imagen conlleva a que el error que pueda tener esta tarea sea sumando al ocasionado por la red neuronal *FaceNet*.

Como líneas futuras, será conveniente probar otro tipo de identificadores, como pueden ser los ofrecidos por *DLib*, junto con otro tipo de detector de rostros basado en redes neuronales, o por OpenCV, basado en Eigenfaces, Fisherfaces y Local Binary Patterns Histograms (LBP). Estos no han sido utilizados en primer lugar porque, según me he documentado, presentaban peores resultados y tiempos de ejecución.

Además, otra tarea futura es la de realizar distintas pruebas sobre otros bancos de imágenes, con un número mayor de estas y rostros que no se encunetren en situaciones tan cercanas, es decir, que no estén directamente sacados de un video, para llegar a una mejor conclusión en lo referido a *FaceNet*.

Finalmente, será de especial interés añadir información corporal, no solo dejando la identificación de una persona ligada tan solo a posibles características que tenga el rostro.

REFERENCIAS

[1] "Alibaba debuts 'smile to pay' facial recognition payments at KFC in China – TechCrunch," [Online]. Available: https://techcrunch.com/2017/09/03/alibaba-debuts-smile-to-pay/?guccounter=1&guce_referrer_us=aHR0cHM6Ly9lcy53aWtpcGVkaWEub3JnLw&guce_referrer_cs=BXFu9xx1J6EEzgdNKueQVg.

[2] P. Viola, M. Jones and others, "Rapid object detection using a boosted cascade of simple features," *CVPR (1),* vol. 1, no. 3, pp. 511-518, 2001.

[3] R. Schapire and Y. Freund, "A decision-theoretic generalization of on-line learning and an application to boosting," in *Second European Conference on Computational Learning Theory*, 1995, pp. 23 -- 37.

[4] T. Ahonen, A. Hadid and M. Pietik{\"a}inen, "Face recognition with local binary patterns," in *European conference on computer vision*, 2004, pp. 469--481.

[5] I. Kukenys and B. McCane, "Support vector machines for human face detection," in *Proceedings of the New Zealand computer science research student conference*, Citeseer, 2008, pp. 226--229.

[6] X. Wang, T. X. Han and S. Yan, "An HOG-LBP human detector with partial occlusion handling," in *2009 IEEE 12th international conference on computer vision*, IEEE, 2009, pp. 32--39.

[7] J. a. Z. G. Xiang, "Joint Face Detection and Facial Expression Recognition with MTCNN," in *2017 4th International Conference on Information Science and Control Engineering (ICISCE)*, 2017, pp. 424--427.

[8] T. Matthew and P. Alex, "Eigenfaces for recognition," *Journal of cognitive neuroscience,* vol. 3, no. 1, pp. 71--86, 1991.

[9] P. N. Belhumeur, J. P. Hespanha and D. J. Kriegman, "Eigenfaces vs. fisherfaces: Recognition using class specific linear projection," *IEEE Transactions on Pattern Analysis \& Machine Intelligence,* no. 7, pp. 711--720, 1997.

[10] N. a. T. B. Dalal, "Histograms of oriented gradients for human detection," 2005.

[11] T. a. P. M. a. H. D. Ojala, "Performance evaluation of texture measures with classification based on Kullback discrimination of distributions," in *Proceedings of 12th International Conference on Pattern Recognition*, IEEE, 1994, pp. 582--585.

[12] T. a. P. M. a. H. D. Ojala, "A comparative study of texture measures with classification based on featured distributions," *Pattern recognition,* vol. 29, no. 1, pp. 51--59, 1996.

[13] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry and Y. Ma, "Robust face recognition via sparse representation," *IEEE transactions on pattern analysis and machine intelligence,* vol. 31, no. 2, pp. 210--227, 2008.

[14] J. V. Davis, B. Kulis, P. Jain, S. Sra and I. S. Dhillon, "Information-theoretic metric learning," in *Proceedings of the 24th international conference on Machine learning*, ACM, 2007, pp. 209--216.

[15] F. Schroff, D. Kalenichenko and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815--823.

[16] P. J. Phillips, P. J. Flynn, T. Scruggs, K. W. Bowyer, J. Chang, K. Hoffman, J. Marques, J. Min and W. Worek, "Overview of the face recognition grand challenge," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, IEEE, 2005, pp. 947--954.

[17] "Chanstk/FaceRecognition_MTCNN_FaceNet," [Online]. Available: https://github.com/Chanstk/FaceRecognition_MTCNN_FaceNet. [Accessed 15 6 2019].

[18] "davidsandberg/facenet," 2019. [Online]. Available: https://github.com/davidsandberg/facenet. [Accessed 15 6 2019].

[19] "CMake," [Online]. Available: https://cmake.org/. [Accessed 15 10 2019].

[20] "git --todo-es-local," [Online]. Available: https://git-scm.com/. [Accessed 15 10 2019].

[21] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097--1105.

[22] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychological review,* p. 386, 1958.

[23] Y. a. B. L. a. B. Y. a. H. P. a. o. LeCun, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, no. 11, pp. 2278--2324, 1998.

[24] D. C. a. M. U. a. M. J. a. G. L. M. a. S. J. Ciresan, "Flexible, high performance convolutional neural networks for image classification," in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[25] L. Wolf, T. Hassner and I. Maoz, Face recognition in unconstrained videos with matched background similarity, IEEE, 2011.

[26] G. B. Huang, M. Mattar, T. Berg and E. Learned-Miller, Labeled faces in the wild: A database forstudying face recognition in unconstrained environments, 2008.

[27] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, Springer, 2014, pp. 818--833.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer*

vision and pattern recognition, 2015, pp. 1--9.

[29] "Inteligencia artificial," 23 06 2019. [Online]. Available: https://es.wikipedia.org/wiki/Inteligencia_artificial. [Accessed 25 06 2019].

[30] "Machine Learning y Deep Learning: aprende las diferencias," 2019. [Online]. Available: https://www.salesforce.com/mx/blog/2018/7/Machine-Learning-y-Deep-Learning-aprende-las-diferencias.html. [Accessed 25 06 2019].

[31] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics,* vol. 36, no. 4, pp. 193-202, 1980.

[32] "Redes neuronales convolucionales - Wikipedia, la enciclopedia libre," 7 8 2019. [Online]. Available: https://es.wikipedia.org/wiki/Redes_neuronales_convolucionales. [Accessed 17 8 2019].

[33] Na8, "¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador," 06 04 2019. [Online]. Available: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/. [Accessed 17 8 2019].

[34] M. Stewart, "Simple Introduction to Convolutional Neural Networks - Towards Data Science," [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed 15 8 2019].

[35] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way -- Towards Data Science," [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Accessed 14 08 2019].

[36] "TensorFlow | TensorFlow Core," [Online]. Available: https://www.tensorflow.org/overview. [Accessed 21 8 2019].

[37] P. Viola, M. J. Jones and D. Snow, "Detecting pedestrians using patterns of motion and appearance," *International Journal of Computer Vision,* vol. 63, no. 2, pp. 153--161, 2005.

[38] Q. Zhu, M.-C. Yeh, K.-T. Cheng and S. Avidan, "Fast human detection using a cascade of histograms of oriented gradients," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, IEEE, 2006, pp. 1491--1498.

[39] M. M. Abdelwahab, S. A. Aly and I. Yousry, "Efficient web-based facial recognition system employing 2DHOG," *arXiv preprint arXiv:1202.2449,* 2012.

[40] "Git," [Online]. Available: https://es.wikipedia.org/wiki/Git. [Accessed 15 10 2019].

[41] T. F. Cootes, G. J. Edwards and C. J. Taylor, "Active appearance models," in *European conference on computer vision*, Springer, 1998, pp. 484--498.