UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo de Fin de Grado

FacesDetector: Aplicación práctica de machine learning sobre imágenes para un contexto de seguridad

Cristian Domínguez Gómez

Julio, 2018

UNIVERSIDAD DE EXTREMADURA

Escuela Politécnica

Grado en Ingeniería Informática en Ingeniería del Software

Trabajo de Fin de Grado

FacesDetector: Aplicación práctica de machine learning sobre imágenes para un contexto de seguridad

Autor: Cristian Domínguez Gómez
Fdo:

Tutor: Pedro José Clemente Martín
Fdo:

Cotutor: Jorge Perianez Pascual
Fdo:

Tribunal Calificador

Presidente: Juan Hernández Núñez
Fdo:

Secretario: Álvaro Prieto Ramos
Fdo:

Vocal: Roberto Rodríguez Echeverría
Fdo:

Índice general

Índice de figuras

Capítulo 1

Introducción

La velocidad con la que avanza la tecnología en los últimos años es cada vez mayor. Lo que hace 10 años era ciencia ficción, ahora es una realidad. Estos cambios quedan reflejados, por ejemplo, en los sistemas de seguridad biométricos. Estos escáneres están presentes en la vida cotidiana, mediante complejos sistemas que son transparentes para el usuario. Gracias a estos avances tecnológicos, sobre todo en potencia de cálculo, casi cualquier dispositivo móvil moderno puede implementar un sistema de desbloqueo mediante huella dactilar casi instantáneo.

Otro ejemplo de seguridad biométrica es el reconocimiento facial. La conocida compañía norteamericana Apple, entre otras, ha implementado en los últimos modelos de su conocido dispositivo móvil el llamado FaceID. Este sistema no es más que un escáner facial, mediante el cual, el usuario puede desbloquear su teléfono móvil tan solo con mirar a la cámara frontal del dispositivo. Si no fuera por los avances en la potencia de cálculo de los procesadores, e incluso de las unidades de procesamiento gráfico, a día de hoy no sería posible disponer de un sistema tan complejo, que a la vez parece tan básico y útil, en un terminal móvil.

Como ya se ha comentado, actualmente existe una buena potencia de cálculo en los sistemas cotidianos actuales. Esto se traduce en la implementación de sistemas cada vez mejores y más complejos. Un claro ejemplo de estas implementaciones es el reconocimiento facial mediante *Machine Learning* o Deep Learning, haciendo uso de redes neuronales.

Aunque no todo es implementar sistemas de seguridad biométricos en dispositivos móviles. Los computadores siguen teniendo una mayor potencia de cálculo para aprovechar los complejos algoritmos de aprendizaje automático o profundo. Explotando los avances mencionados y aplicándolos a a sistemas de aprendizaje profundo, existen desarrollos en seguridad biométrica, como el reconocimiento facial, para entornos empresariales, aunque estos desarrollos tienen un problema: la mayoría de estos sistemas son de pago.

La comunidad de desarrollo de este tipo de sistemas cada vez gana más fuerza pero no todos los proyectos se adaptan a las necesidades personales de cada individuo o de cada entorno empresarial. Algunos proyectos se realizan de forma concreta para un determinado cliente, otros son gratuitos pero no son *open-source*, algunos se quedan sin soporte... Aunque existen algunos sistemas competentes, otros muchos no se ajustan a los tiempos actuales, en los que el proceso de digitalización avanza dando pasos de gigante.

Como conclusión, se puede obtener que ahora que la potencia de cálculo de los procesadores modernos es tan grande, como se ha mencionado ya varias veces, es posible desarrollar un sistema de seguridad biométrico como el descrito, aplicando técnicas complejas de aprendizaje automático y de *Deep Learning* o redes neuronales.

Resumen

En el presente documento se presenta *FacesDetector*, una aplicación práctica de *Machine Learning* sobre imágenes para un contexto de seguridad. El objetivo principal de este Trabajo de Fin de Grado es desarrollar un sistema de reconocimiento facial que se pueda aplicar en cualquier entorno, tanto personal como empresarial, en materia de seguridad.

Para ilustrar el objetivo de este proyecto, un ejemplo de uso concreto para el sistema desarrollado es el del laboratorio de Quercus, situado en el edificio de investigación. Actualmente, el sistema existente para entrar en el laboratorio es mediante una tarjeta NFC que, al situarla sobre la cerradura, ésta abre la puerta. Este sistema se podría sustituir o complementar con este Trabajo de Fin de Grado. Simplemente sería necesario colocar una cámara IP sobre la puerta y que, al detectar el rostro de una persona, compruebe si está autorizada para entrar al laboratorio. En caso de que esté autorizada, se activaría el mecanismo de apertura ya existente de la cerradura de la puerta.

En caso contrario, simplemente, no se abriría la puerta.

El proyecto podrá ser desplegado en cualquier contexto, ya sea personal o empresarial, como ya se ha comentado. Respondiendo al problema sobre la falta de proyectos *open-source* de este tipo expuesto en la introducción del presente documento, junto a éste se entrega todo el código desarrollado y, además, también se publicará el código en la plataforma Github [1] para que cualquier persona pueda reutilizar este sistema de forma gratuita y editar el código en función de las necesidades de cada usuario. La dirección de este repositorio es https://github.com/smackcristian/tfg

Este documento se encuentra dividido en cinco secciones, además de la actual: en primer lugar, en el siguiente capítulo, se analizará el estado de arte, explicando los conceptos de *Machine Learning* y *Deep Learning* mencionados anteriormente, entre otros, tan utilizados en los últimos tiempos. En los capítulos posteriores, se describirá la propuesta de una solución, cumpliendo los objetivos definidos anteriormente; la validación tecnológica del sistema y posibles trabajos futuros que no han llegado a implementarse.

Capítulo 2

Estado del arte

En este apartado se describen los conceptos de Machine Learning y Deep Learning, así como algunos de los sistemas actuales en el mercado y las alternativas tecnológicas que existen para el desarrollo de un sistema similar.

2.1. Machine Learning

Se trata de una disciplina científica relacionada con la Inteligencia Artificial. Su principal objetivo es crear sistemas que *aprendan* automáticamente, es decir, que sean capaces de identificar patrones complejos en un gran volumen de datos que reciba dicho sistema. Esta disciplina surgió en los años 60, producto de las ciencias de la computación, influenciada por la neurociencia.[2]

Con el paso de los años, esta disciplina fue enfocándose en diferentes asuntos, como el razonamiento probabilístico, predicción del tiempo, predicción de ventas... Al principio, esta disciplina tenía unas funciones bastante básicas y limitadas, aunque en los últimos años se utiliza para casos tan complejos como detección de cáncer o predicción del tráfico.[3]

Los algoritmos con los que trabaja esta disciplina necesitan de una gran cantidad de datos para poder *entrenar* el sistema. Estos algoritmos se pueden clasificar en tres grandes campos, los cuales son:

- ***Supervised learning* o aprendizaje supervisado**: estos tipos de algoritmos dependen de datos previamente etiquetados, para que, por ejemplo, un computador sea capaz de distinguir imágenes de vehículos. Lo normal, para aprovechar la calidad de los datos es que su categoría sea previamente etiquetada por personas. Parte de problemas ya resueltos pero que podrían darse en un futuro. Un ejemplo es el reconocimiento de manuscritos.
- ***Unsupervised learning* o aprendizaje sin supervisión**: en este grupo, a diferencia del anterior, los datos no necesitan de etiqueta alguna. En su lugar se le ofrece una gran cantidad de características de los datos y el sistema debe ser capaz de encontrar la estructura existente. Un ejemplo de aprendizaje sin supervisión es el *clustering*, que se encarga de agrupar datos según ciertas características, por ejemplo, para poder reconocer expresiones faciales, como si una persona sonríe.
- ***Reinforcement Learning* o aprendizaje por refuerzo**: este tipo de aprendizaje se basa en el aprendizaje en base a premios, a través de prueba y error. Los datos son obtenidos del entorno. Si la máquina realiza una acción acertada, se premia con valores positivos para encontrar una solución óptima.

A día de hoy existen multitud de casos de uso de Machine Learning, que son tan transparentes y cotidianos para los usuarios que pasan casi desapercibidos para muchos de ellos. Algunos ejemplos de estos casos de uso son los siguientes:

- Procesamiento del lenguaje natural (en texto).
- Predicción del tiempo.
- Clasificación de Emails.
- Reconocimiento de manuscritos.
- Diagnóstico médico.

2.2. Deep Learning

Otra disciplina, contenida en el ámbito del Machine Learning es el Deep Learning. Esto es el mismo proceso que realiza el Machine Learning, pero utilizando una red neuronal artificial.[4] Es decir, se utilizan estructuras lógicas que se asemejan en mayor medida a la organización del sistema nervioso de los mamíferos, especializándose en detectar determinadas características existentes en los objetos o datos percibidos.

Una red neuronal estándar se basa en muchos procesadores simples y conectados entre ellos llamados neuronas, cada una de ellas produciendo una secuencia de activaciones con valores reales. Las neuronas de entrada se activan en base a los datos que actúan como flujo de entrada. Otras neuronas se activan a través de conexiones ponderadas desde neuronas previamente activas.

El aprendizaje en las redes neuronales consiste en encontrar pesos que hagan que la red presente el comportamiento deseado. Estos pesos asignados a cada neurona de la red se van modificando hasta que se encuentra el adecuado. Según el tamaño de la red y el conjunto de datos de entrada, puede requerir un mayor entrenamiento, donde se van modificando estos pesos mencionados anteriormente.[5]

El principal objetivo de las redes neuronales es encontrar patrones de comportamiento en los datos de entrada para producir una determinada salida, independientemente del tipo de red neuronal que sea. Existen multitud de tipos de redes neuronales, como las redes convolucionales, utilizadas sobre todo para la búsqueda de patrones en imágenes en base a unas complejas operaciones matemáticas para la posterior clasificación de las mismas, o las redes neuronales *feedforward*, siendo éstas las más simples. El uso de una u otra arquitectura depende del objetivo para el cual se diseña la red.

El uso de esta disciplina está creciendo, pues en este momento, se puede aprovechar el potencial del Deep Learning gracias a la potencia del cálculo que brindan los procesadores modernos y las GPUs, sin necesidad de disponer de la última tecnología disponible en el mercado. Para reafirmar lo mencionado, existen casos de usos reales donde se están aplicando redes neuronales como técnica de aprendizaje automático. Algunos ejemplos de casos de uso de redes neuronales o Deep Learning son[6]:

- Reconocimiento de voz.

- Detección de movimiento.
- Reconocimiento facial.
- Búsqueda de imágenes.
- Búsqueda por voz.
- Detección de fraudes.

A modo de resumen de los dos conceptos anteriores, relacionados entre ellos, en la siguiente figura [2.1] se puede observar cómo están relacionados los conceptos de Inteligencia Artificial, Machine Learning y Deep Learning entre sí, siendo partes unos de otros.

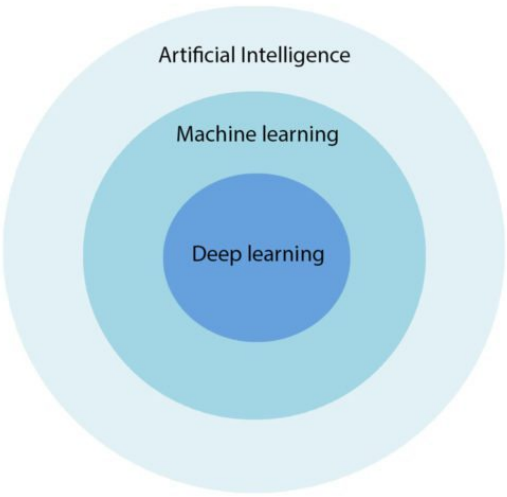

Figura 2.1: Relación entre Inteligencia Artificial, Machine Learning y Deep Learning

2.3. Sistemas actuales en el mercado

Los sistemas actuales que se pueden encontrar, habitualmente trabajan con redes neuronales, ya que su uso está creciendo, aunque también se pueden encontrar proyectos que no precisan de las mismas, utilizando simplemente algoritmos de Machine Learning.

Realizando diferentes búsquedas en Internet, se han podido encontrar diferentes proyectos, la mayoría en Github, enfocados al reconocimiento facial. Unos utilizan redes convolucionales, mientras que otros no lo ven necesario y aplican técnicas de Machine Learning. Algunos de los proyectos, tanto comerciales como no, que se encuentran basados en Deep Learning son los siguientes:

- **FaceNet**: según sus creadores [7], se trata de un sistema para verificación, reconocimiento y detección facial. Utilizan redes neuronales previamente entrenadas para optimizar el reconocimiento. Estas redes realizan operaciones convolucionales en busca de patrones en la imagen tomada. Es un proyecto de Google.
- **OpenFace:** se trata de un proyecto *open-source* que es una implementación FaceNet, utilizando también redes neuronales. Este proyecto está escrito en Python, utilizando Torch [8], que es un framework para el desarrollo de redes neuronales, que se detallará más adelante. Al estar basado en FaceNet, este proyecto es capaz de verificar si dos fotografías pertenecen a la misma persona, detectar caras y reconocimiento de las mismas.
- **DeepFace[9]:** este proyecto ha sido desarrollado por un equipo de Facebook. Cuenta con 9 capas, entrenadas con los datos de los usuarios de la red social. Ofrece un 97 % de precisión.
- **DeepSight[10]:** desarrollado por la empresa hindú BaseApp. Hace uso también de redes neuronales para la detección de caras. El SDK que ofrece esta empresa también cuenta con una utilidad para el reconocimiento facial de personas.
- **FaceAPI[11]:** desarrollado por Microsoft. Es una herramienta bastante potente porque, además de reconocer personas es capaz de detectar las emociones de las mismas. Aunque es de pago, permite realizar algunas pruebas a partir de su página web.

Como se puede observar, existen multitud de proyectos *open-source* y privados que utilizan Deep Learning para la detección y el reconocimiento facial, incluso realizando una rápida búsqueda en Github se pueden encontrar diferentes implementaciones en distintos lenguajes de programación.

Muchos de estos proyectos tienen bastantes características en común en cuanto a su implementación. Entre ellas están el almacenamiento de datos para poder recuperar la información para entrenar una red o, simplemente, para comparar dos rostros, partiendo de una imagen almacenada en base de datos. También, para su desarrollo, se habrá utilizado un framework o alguna librería destinada al Deep Learning.

Al dedicarse al reconocimiento o la detección facial también deben utilizar algún sistema para poder procesar las imágenes correctamente, los flujos de cámara... y, además, algunas herramientas que utilizan modelos de Deep Learning, ya sea para reconocimiento facial o no, también hacen uso de APIs para que los usuarios ajenos al sistema puedan utilizar los servicios, como es el caso de Face API de Microsoft[11].

Por ello, en el siguiente apartado de este documento se analizarán algunas de las diferentes alternativas tecnológicas existentes actualmente para poder realizar un desarrollo de un sistema de reconocimiento facial aplicando Deep Learning.

2.4. Alternativas tecnológicas

Como se ha mencionado anteriormente, en esta sección se analizan diferentes tecnologías destinadas al almacenamiento de datos, desarrollo de modelos de Deep Learning, API Rest y visión por computador para el desarrollo del sistema de reconocimiento facial descrito en la introducción del presente documento.

2.4.1. Almacenamiento de datos

Es necesario almacenar en algún lugar los datos extraídos de las características faciales de una persona para, posteriormente, poder trabajar con

ellos y entrenar modelos. A continuación se muestran diferentes alternativas enfocadas a esta labor de almacenamiento persistente de datos en el sistema.

MySQL

Es el Sistema Gestor de Bases de Datos más popular. Este SGBD es propiedad de Oracle Corporation y se distribuye en dos versiones: *Community Edition*, que es *open-source* y la versión de pago, que es la *Standard Edition*. También cuenta con una versión realizada por la comunidad que es MariaDB.

La instalación de MySQL es relativamente sencilla y, además, no exige demasiados recursos. Por otro lado, el lenguaje SQL utilizado para este SGBD, presenta ciertas diferencias con el estándar.

Las ventajas de MySQL son las siguientes:

- Al ser uno de los SGBD más extendidos, existe una gran cantidad de documentación desponible.
- Al ser de los más extendidos, es muy probable encontrar fácilmente la solución a un problema concreto.
- Compatible con los sistemas operativos más habituales.
- Existe un proyecto *open-source* de MySQL llamado MariaDB.

Las desventajas son:

- No toda su funcionalidad es de código abierto.
- Tiene diferencias respecto al estándar ANSI SQL[12].
- No soporta transacciónes y es propenso a la corrupción de los datos.[13]

Oracle Database

Es otro SGBD de tipo objeto-relacional. Cuenta con diferentes versiones, pero la única gratuita es la *Express Edition*. Este Sistema Gestor de Bases de Datos está muy enfocado al ámbito empresarial y, en la práctica, su uso está muy extendido también.

Es un SGBD multiplataforma, pues es capaz de correr en diferentes sistemas, y permite la partición de memoria para mejorar la eficiencia del sistema, lo que se traduce en una gran escalabilidad para este sistema gestor de base de datos.

Las ventajas con las que cuenta este SGBD son:

- Cuenta con una versión gratuita con multitud de utilidades.
- Implementa el estándar ANSI SQL de 2012.[14]
- Está muy extendido.[15]
- Es escalable.
- Soporta PL/SQL lo que lo hace muy completo.

Las desventajas que presenta Oracle Database son:

- La instalación es algo más tediosa que otros SGBD.
- Las versiones completas son de pago.
- Las licencias son algo caras [15]

Serialización de objetos

La serialización de un objeto consiste en generar una secuencia de bytes a partir de la información del mismo, para que pueda ser almacenado o transmitido. Existen varios lenguajes que soportan la serialización de objetos. Por ejemplo, en Java para que un objeto se pueda serializar, debe implementar la interfaz *serializable*, mientras que en Python, se hace uso de una librería llamada *pickle*, ya que, por particularidades de este lenguaje, cualquier objeto se puede serializar. Este objeto serializado se puede volcar a un fichero desde el que se puede volver a recuperar realizando la operación inversa a serializar, que es deserializar.

La serialización de objetos cuenta con una serie de ventajas:

- No requiere de software externo.
- Es muy sencillo e intuitivo de utilizar.
- La gran mayoría de lenguajes de programación de alto nivel lo soportan.
- Los ficheros serializados pesan menos que una base de datos, ya que, por ejemplo, no almacenan índices.
- Por lo general, los ficheros con objetos serializados no suelen pesar demasiado.
- Se puede recuperar la información sin necesidad de implementar ningún conector.

Por otro lado, también tiene algunas desventajas:

- No es legible para los humanos.
- En caso de utilizar objetos grandes, puede ser un poco lento.

Tecnología	**Ventajas**	**Desventajas**
MySQL	Mucha documentación. Gran Comunidad. Compatible con los SO más actuales. A pesar de no ser 100 % abierto, existe una versión de la comunidad que sí lo es llamada MariaDB	No es 100 % código abierto. No soporta transacciones. Tiene diferencias respecto al estándar ANSI SQL.
Oracle Database	Cuenta con una versión gratuita.Está muy extendido.Soporta PL/SQL. Escalable	Instalación tediosa. Versiones completas de pago.
Serialización de objetos	No requiere de software externo.Muy sencillo e intuitivo de utilizar. Aceptado por muchos lenguajes. Ficheros no muy grandes.	No legible para los humanos. En caso de utilizar objetos grandes, puede llegar a ser un poco lento.

Cuadro 2.1: Tabla resumen. Alternativas tecnológicas para almacenamiento de datos

2.4.2. Librerías y frameworks para Deep Learning

Para trabajar con redes neuronales existen multitud de herramientas disponibles en el mercado, algunas más modernas que otras y escritas en distintos lenguajes de programación. En este apartado se describirán algunas de las librerías y frameworks para el desarrollo de redes neuronales.

Caffe

Sus siglas significan *Arquitectura Convolucional para Incorporación Rápida de Características.*[16] Es un framework para Deep Learning, desarrollado en la Universidad de California, Berkeley. Se trata de una herramienta *open-source*, bajo una licencia BSD. Está escrito en C++ con una interfaz Python. También soporta varias tecnologías, entre ellas CUDA.[17]

La característica más interesante de Caffe es su velocidad. Según el sitio web del proyecto [16], es capaz de procesar 60 millones de imágenes por día en una sola NVIDIA K40 y que, además, es el más rápido para redes convolucionales.

Las ventajas de este framework para Deep Learning son las siguientes:

- Rápido procesamiento de imágenes.
- Interfaz para Python y Matlab.
- Soporte para entrenamiento sobre GPU (Graphics Processing Unit).

Las desventajas de este framework son:

- Fuera de las redes convolucionales, la usabilidad cae.
- Solamente tiene un formato de salida: HDF5, un formato de fichero utilizado para almacenar un volumen grande de datos y poder gestionarlos fácilmente.[18]

Dlib

Dlib[19] es un conjunto de herramientas en C++ que contienen diferentes algoritmos de Machine Learning y Deep Learning para crear software complejo en C++ para resolver problemas del mundo real.

Según sus desarrolladores[19] también se puede utilizar para procesado de imágenes. Cuenta con un modelo preentrenado para la detección facial en base a 68 marcas faciales, entre las que se encuentran las cejas, la boca... Esta librería es gratuita y *open-source*, por lo que se puede utilizar en cualquier aplicación. Se utiliza en multitud de dominios, desde robótica hasta teléfonos móviles.

Las ventajas de Dlib son:

- Es gratuita y de código abierto.
- Python tiene soporte para sus modelos.
- Tiene un modelo preentrenado para detección facial.

Las desventajas de esta librería son:

- Si no se tiene un alto conocimiento de C++ puede ser tedioso editar el código.
- El soporte por parte de la comunidad no es tan grande como el de otras herramientas.

TensorFlow

TensorFlow[20] es una librería *open-source* para computación númerica de alto rendimiento. Cuenta con un buen soporte tanto para Machine Learning como para Deep Learning. Gracias a su flexibilidad, también se puede extrapolar a otros ámbitos científicos.

Esta potente herramienta para Deep Learning ha sido desarrollada por Google y soporta diferentes lenguajes, entre los que están Python, Java o C++.

Soporta CUDA, dando soporte únicamente a GPUs de la firma NVIDIA y, además, se puede separar la forma de ejecución, sea en CPU o GPU, en dos paquetes de instalación diferentes.

Las ventajas e inconvenientes que tiene este framework de desarrollo para Deep Learning desarrollado por los trabajadores de Google, se enumeran a continuación.

Ventajas:

- Escalabilidad.
- Se puede paralelizar muy bien.
- Soporta varios lenguajes.
- Puede ejecutarse a través de Docker.
- Está extendido, se pueden encontrar multitud de proyectos hechos con este framework.

Inconvenientes:

- Tiene una curva de aprendizaje grande.
- El diseño de las redes es algo tedioso.
- Solamente ofrece soporte a GPUs NVIDIA.
- A veces, las actualizaciones rompen la retrocompatibilidad.

Theano

Theano[21] es una librería para Python que permite definir, optimizar y evaluar expresiones matemáticas que implican arrays multidimensionales de forma eficiente. Tiene integración con la librería *numpy* de Python. Es capaz de trabajar con GPU de forma transparente y presenta generación dinámica de código C, lo que hace que las expresiones se evalúen de una forma más rápida.

Algunas de las ventajas con las que cuenta Theano se enumeran a continuación:

- La flexibilidad que ofrece.
- Si se usa correctamente, su eficiencia.

Theano también cuenta con inconvenientes, algunos de ellos son:

- Ya no tiene mantenimiento.
- Curva de aprendizaje grande.
- A veces puede ser lento.

Keras

Keras[22] es una API de alto nivel para Deep Learning. Está escrita en Python y es capaz de ejecutarse sobre Tensorflow, Microsoft Cognitive Toolkit o Theano para simplificar el desarrollo de redes neuronales en estos frameworks. Fue desarrollada enfocándose en la posibilidad de una experimentación rápida con redes neuronales. Permite un rápido y fácil prototipado de redes, soportando tanto redes convolucionales y recurrentes, así como una mezcla de ambas.

Una de las características más interesantes de Keras es la posibilidad de ejecutarse sobre CPU o GPU sin tener que especificarlo, ya que viene indicado por la herramienta que va por debajo. A pesar de ser un framework para simplificar el desarrollo de redes neuronales en Tensorflow, por ejemplo, también cuenta con una serie tanto de ventajas como de inconvenientes. Estos se enumeran a continuación.

Las ventajas propias de esta herramienta son:

- Desarrollo rápido de modelos.
- Curva de aprendizaje pequeña.
- Puede ejecutarse utilizando diferentes frameworks que hacen de motor.
- Una comunidad grande de desarrolladores.

Las desventajas de este framework son:

- Si se desea desarrollar algo más específico, es muy difícil con Keras.
- Solamente tiene un formato de salida para los pesos: HDF5.[18]

Torch

Torch es un framework de computación científica con un amplio soporte para algoritmos de Machine Learning que corre sobre GPUs. Según la página de la herramienta, es fácil de usar y eficiente, gracias LuaJIT, un lenguaje de *scripting* fácil y rápido. También soporta CUDA y redes neuronales.

Según sus desarrolladores [8], cuenta con una amplia gama de características, entre las que se pueden encontrar múltiples rutinas para indexación, rutinas de álgebra lineal... Quizá la característica más interesante de este framework es que cuenta con *ports* para iOS y Android.

Las ventajas que presenta Torch son:

- Combinado con Pytorch puede utilizarse con Python.
- Tiene cantidad de modelos preentrenados.

Sus inconvenientes son:

- Está escrito en Lua, por lo que hay que aprender un nuevo lenguaje.
- No tiene mucha documentación.

En la tabla que aparece a continuación [2.4.2] se recogen las ventajas y desventajas, a modo de resumen, de todas las alternativas tecnológicas citadas anteriormente destinadas al desarrollo de modelos de Deep Learning.

Tecnología	**Ventajas**	Desventajas
Caffe	Procesamiento rápido de imágenes. Interfaz para Python y Matlab. Soporte para entrenamiento sobre GPU	La usabilidad cae sin redes neuronales convolucionales
Dlib	Es gratuita y de código abierto. Python soporta sus modelos. Tiene un modelo preentrenado para detección facial.	Con bajos conocimientos de C++ puede ser complicado editar el código. El soporte por parte de la comunidad no es tan grande.
Tensorflow	Escalabilidad. Se puede paralelizar bien. Soporta varios lenguajes. Puede ejecutarse a través de Docker. Está muy extendido	Curva de aprendizaje grande. Diseño de las redes tedioso. Solo soporta GPUs Nvidia. Las actualizaciones a veces rompen la retrocompatibilidad.
Theano	Flexibilidad. Usado correctamente, su eficiencia	Ya no tiene mantenimiento. Curva de aprendizaje grande. A veces puede ser lento
Keras	Desarrollo rápido. Curva de aprendizaje pequeña. Puede ejecutarse utilizando diferentes frameworks. Gran comunidad	Complicado para algo muy concreto. Solamente un formato de salida para los pesos.
Torch	Puede usarse con Python junto a Pytorch. Multitud de modelos preentrenados.	Escrito en Lua. No tiene mucha documentación

Cuadro 2.2: Tabla Resumen. Frameworks y librerías para Deep Learning

2.4.3. Herramientas para visión por computador

En este apartado se comentarán las diferentes herramientas encontradas para tratar el procesamiento de imágenes a través de un computador, ya sea de una fotografía o directamente un flujo de vídeo, también conocido como *Computer Vision* o visión por computador.

OpenCV

OpenCV[23] es una herramienta *open-source* publicada bajo licencia BSD. Es multiplataforma, pues soporta Windows, Linux, Mac OS, iOS y Android. Cuenta con interfaces Python, C++ y Java, por lo que es bastante completo. Fue diseñado para eficiencia computacional, enfocándose en aplicaciones en tiempo real. Está escrita en C/C++ y soporta procesamiento multi-núcleo. Con OpenCL, puede tomar las ventajas de la aceleración de hardware.

Su uso está muy extendido, con una gran comunidad de usuarios. Los usos van desde el arte interactivo, a la inspección de minas o robótica avanzada. También se utiliza para detección facial y de diferentes objetos del entorno, como el proyecto YOLO[24], que hace uso de OpenCV.

Esta herramienta tiene bastantes ventajas respecto a otros software para visión por computador. Como todo, también tiene una serie de inconvenientes.

Ventajas:

- Es muy versátil.
- Multiplataforma.
- Es Open-Source.
- Tiene gran cantidad de documentación.

Inconvenientes:

- La instalación es algo complicada.

- A veces, es complicado hacer lo deseado.

SimpleCV

SimpleCV[25] es un framework *open-source* para tratamiento de imágenes digitales. Está hecho sobre OpenCV utilizando Python. Provee una interfaz bastante sencilla para cámaras y manipulación de imágenes, extracción de imágenes y conversión de formato.

Ventajas:

- Instalación sencilla.
- Fácil de usar.
- Está desarrollado sobre OpenCV.

Inconvenientes:

- Actualmente sólo está disponible para Python.
- A veces es complicado hacer lo deseado.

En la siguiente tabla [2.4.3] se recogen, a modo de resumen, las características de las herramientas vistas como alternativas para realizar trabajos de visión por computador.

2.4.4. Herramientas para el desarrollo de API Rest

Existen multitud de frameworks para desarrollar una API Rest de forma rápida y sencilla. En este apartado se enunciarán los frameworks de uso más común para el desarrollo de estas API, cuya finalidad en este proyecto es la posibilidad de utilizar un cliente remoto para la utilización del servicio de reconocimiento facial.

Tecnología	**Ventajas**	Desventajas
OpenCV	Muy versátil. Multiplataforma. Es *open-source*. Mucha documentación	Instalación compleja. A veces, es complicado hacer lo deseado.
Simple CV	Instalación sencilla. Fácil de usar. Desarrollado sobre OpenCV.	Actualmente sólo está disponible para Python. A veces es complicado hacer lo deseado.

Cuadro 2.3: Tabla Resumen. Herramientas para visión por computador

Flask

Flask[26] es un micro-framework escrito en Python que permite un rápido desarrollo de aplicaciones web con un número de líneas de código muy inferior comparado con otros frameworks. Utiliza el motor de *templates* Jinja2 y es distribuido bajo la licencia BSD. También cuenta con un pequeño servidor embebido para utilizar durante el desarrollo.

Las ventajas y desventajas con las que cuenta Flask son las siguientes:

Ventajas:

- Instalación sencilla mediante *pip*, el gestor de dependencias para Python.
- Los *endpoints* se pueden definir rápidamente.
- Es ligero y tiene buen rendimiento.
- Cuenta con un servidor embebido para pruebas.

Inconvenientes:

- Si se quiere usar para realizar una aplicación web completa, es complicado.
- En producción necesita de un servidor como Gunicorn[27].

Express.js

Express[28] es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles. Cuenta con multitud de métodos de utilidad HTTP y middleware para que el desarrollo de la API sea rápido y sólido.

El desarrollo de la API Rest es bastante rápido y sencillo. Instancia un servidor embebido para poder trabajar sobre él, y no necesita de un servidor de aplicaciones externo, como podría ser Tomcat.

Las ventajas de Express.js son:

- Tiene una gran cantidad de librerías.
- Fácil de aprender.
- Cuenta con un servidor propio embebido.

Los inconvenientes que tiene Express.js son:

- Instalación algo compleja con *npm*, un gestor de dependencias para Node.js; por las diferentes versiones que hay.
- Complicado trabajar con bases de datos relacionales.
- No es escalable.

Spring Boot

Spring Boot es una parte de Spring que permite crear diferentes tipos de aplicaciones de una manera rápida y sencilla. Sus características principales son que provee out-of-the-box una serie de elementos que permiten desarrollar diferentes tipos de aplicaciones de forma casi inmediata.

Una de sus características más interesantes es que cuenta con servidores de aplicaciones, como puede ser Tomcat, embebidos. Soporta anotaciones, comúnmente utilizadas para inyección de dependencias o configuración.

Las ventajas de Spring Boot son:

- Desarrollo rápido.
- Gestión de dependencias gracias a Maven.
- Cuenta con un servidor propio embebido.

Sus inconvenientes son:

- El binario, a veces, pesa más porque mete dependencias que no se usan.
- Hay que familiarizarse antes con Spring.
- Puede ser algo más lento que otros frameworks.

Jersey

Jersey[29] es un proyecto que se rige por *JAX-RS*, una API de programación para Java que cuenta con todo lo necesario para el desarrollo de API Rest. Jersey es muy liviano y se puede instalar fácilmente mediante el gestor de dependencias *Apache Maven*, así como ejecutarlo de forma sencilla. Jersey también cuenta con diferentes ventajas e inconvenientes. Algunas de sus ventajas se enumeran a cotinuación:

- No es necesario montar todo un proyecto como Spring.
- Se puede ejecutar en cualquier plataforma que cuente con Java.
- Tiene una gran cantidad de documentación.

Algunos de sus inconvenientes se enuncian a continuación:

- El soporte por parte de la comunidad no está tan extendido como Spring-Boot, por ejemplo.
- El uso del patrón MVC es más pobre que en Spring u otros frameworks.

En la tabla que se muestra a continuación[2.4.4] pueden verse las ventajas y desventajas de las tecnologías utilizadas para la construcción de API Rest que se han visto a lo largo de este apartado

Tecnología	**Ventajas**	Desventajas
Flask	Instalación sencilla. Los *endpoints* se definen rápidamente. Ligero y con buen rendimiento, Servidor embebido para pruebas.	Complicado para aplicación web completa. Necesita de un servidor en producción
Express.js	Gran cantidad de librerías. Fácil de aprender. Servidor embebido.	No es escalable. Instalación algo compleja. Complicado para bases de datos relacionales.
Spring Boot	Desarrollo rápido. Gestión de dependencias gracias a Maven. Servidor embebido	El binario a veces contiene dependencias que no se usan. Hay que familiarizarse con Spring. Puede ser algo más lento que otros frameworks
Jersey	No es necesario montar todo un proyecto como Spring. Se puede ejecutar en cualquier plataforma que cuente con Java. Gran cantidad de documentación.	El uso del patrón MVC es más pobre que en Spring u otros frameworks. No tiene una comunidad tan grande como otros frameworks.

Cuadro 2.4: Tabla Resumen. Herramientas para la construcción de API Rest

Posteriormente se valorarán todas estas alternativas tecnológicas, con sus pros y contras, y se determinará cuáles serán seleccionadas para el desarrollo del proyecto de reconocimiento facial.

2.4.5. Tecnologías seleccionadas

Tras analizar las diferentes tecnologías expuestas anteriormente, para la realización de este trabajo, se han seleccionado las siguientes:

Almacenamiento de Datos	Serialización de objetos
Herramientas para Deep Learning	Keras & Tensorflow
Procesamiento de imágenes	OpenCV & Dlib
API Rest	Flask

Cuadro 2.5: Tabla con las tecnologías seleccionadas

Almacenamiento de datos

Se ha seleccionado la serialización de objetos porque, de esta forma, no se necesita de software externo como un SGBD para el almacenamiento de los datos, pues la mayoría de lenguajes existentes implementan esta característica; y Python. Otro de los motivos es por la facilidad de uso de esta característica y que, además, el tamaño del objeto serializado va a ser menor que si se utilizara un Sistema Gestor de Bases de Datos.

Herramientas para Deep Learning

Para el desarrollo de redes neuronales se han seleccionado Keras, utilizando Tensorflow por debajo. El motivo por el que se ha seleccionado Keras es por la facilidad de desarrollo que da para Deep Learning, ya que se pueden implementar las redes en pocas líneas en el lenguaje de programación Python.

La razón por la que se ha seleccionado Tensorflow es porque, además de que Keras lo puede utilizar, es bastante más potente que otras herramientas del mercado y cuenta con el soporte de Google y de una gran comunidad por detrás. También es el framework más utilizado actualmente para Deep Learning[30]. Además, al instalar Tensorflow para trabajar con Python, se puede seleccionar fácilmente si va a utilizar la CPU o la GPU, aprovechando todo el potencial que da CUDA.

Procesamiento de imágenes

Se ha elegido OpenCV por su gran versatilidad frente a SimpleCV, ya que este último no es más que una especie de *wrapper* de OpenCV.

OpenCV permite cargar los modelos preentrenados de Dlib, los cuáles se utilizarán para detectar rostros humanos en una fotografía o *frame* de un vídeo, además, la curva de aprendizaje de OpenCV no es especialmente grande, ya que soporta Python. También es un punto a su favor la escalabilidad que ofrece este software de visión por computador y la gran comunidad con la que cuenta, ya que cualquier problema que se pudiera presentar durante el desarrollo del proyecto, se podría solucionar fácilmente investigando un poco en Stackoverflow.

API Rest

Flask ha sido la alternativa tecnológica elegida para el desarrollo de una API Rest que permita trabajar desde un cliente externo con el modelo de Deep Learning. La gran ventaja que ofrece este framework es que el tiempo empleado en el desarrollo de la API es ínfimo, contando, además, con toda la versatilidad que ofrece Python.

Otro de los motivos por los que este micro-framework ha sido elegido es por la sencillez a la hora de definir sus *endpoints*, a diferencia de los mencionados anteriormente en el análisis de las alternativas tecnológicas. Su escalabilidad es el otro punto fuerte de este framework, así como la facilidad de integrar librerías.

Capítulo 3

Propuesta de Solución

En este apartado de la presente documentación se detalla la propuesta de solución del proyecto. Esta propuesta va desde el análisis y diseño del proyecto hasta las pruebas del mismo una vez desarrollado.

3.1. Análisis y Diseño

A continuación se exponen los diferentes requisitos del proyecto, tanto los requisitos funcionales como los no funcionales y las restricciones. También se incluirán un diagrama reflejando los casos de uso del sistema a desarrollar y un diagrama de secuencia de cada uno de los diferentes casos de uso reflejados en el diagrama mencionado anteriormente.

3.1.1. Requisitos funcionales del proyecto

En este apartado se enumeran los requisitos funcionales que debe cumplir el proyecto:

- El sistema deberá reconocer a diferentes personas, tanto desconocidas como no desconocidas.

- A las personas desconocidas se le debe mostrar una etiqueta que la identifique como *Desconocido*.
- El sistema deberá permitir el entrenamiento con nuevas personas.
- El sistema deberá contar con un servicio intermedio desde el cual cualquier cliente desarrollado pueda utilizar los diferentes métodos del sistema desarrollado, por ejemplo, por medio de una API.
- El sistema debe ser capaz de procesar, tanto fotografías de la cara de una persona, como vídeos.
- El sistema debe ser capaz de tomar tanto el flujo de una cámara IP como de una *webcam* conectada al equipo donde se despliegue.
- Se debe desarrollar un cliente que pueda enviar fotos y vídeos al sistema, así como entrenarlo y procesar los vídeos.
- El sistema será desarrollado utilizando redes neuronales.

3.1.2. Requisitos no funcionales del proyecto

En esta sección del documento se listan los diferentes requisitos no funcionales del software a desarrollar:

- El software desarrollado podrá ser desplegado en cualquier equipo de escritorio o portátil con al menos un procesador de la serie *Core i* de Intel, aunque si es de los últimos modelos y, además, cuenta con una GPU dedicada, el sistema trabajará más rápido.
- El sistema deberá guardar los mejores resultados obtenidos hasta el momento durante su entrenamiento, de forma que pueda cancelarse en cualquier momento.
- El sistema debe ser escalable.
- Respecto al servicio intermedio, debe controlar el acceso concurrente a la hora de entrenar el sistema o procesar vídeos para evitar errores.
- El cliente desarrollado podría ser un pequeño bot de Telegram.

3.1.3. Requisitos de información

En este apartado se mencionan los requisitos de información correspondientes al software necesario para el proyecto:

- Deben almacenarse tanto las imágenes, como los vídeos correspondientes a cada persona, así como un identificador de la misma, que puede ser, por ejemplo, su nombre.
- Deben almacenarse los pesos de la red neuronal una vez haya sido entrenada.

3.1.4. Restricciones

En esta sección del documento se enumeran las restricciones impuestas al proyecto:

- Debe desarrollarse, al menos el sistema de reconocimiento facial, con Python.
- Desde el cliente que se desarrolle, únicamente pueden entrenar la red y procesar los vídeos un determinado número de usuarios.

3.1.5. Diagrama de casos de uso

A continuación se incluye un diagrama con los casos de usos que se consideran necesarios y se corresponden al sistema a desarrollar tras un estudio de los requisitos del mismo:

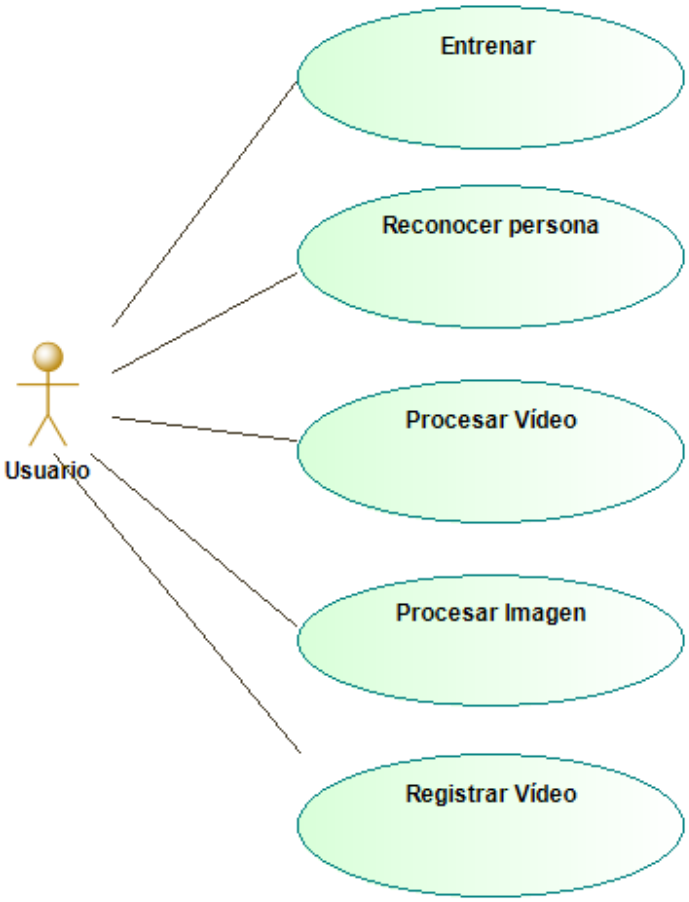

Figura 3.1: Diagrama de casos de uso

Como se puede observar en el diagrama anterior, el sistema contará con cuatro casos de uso:

- **Entrenar**: se entrenará la red neuronal con las personas disponibles.
- **Reconocer persona**: dada una imagen o flujo de cámara, el sistema será capaz de reconocer a las personas detectadas.
- **Procesar videos**: recibido un vídeo de una persona, será capaz de procesar los datos necesarios
- **Procesar imágenes**: el sistema será capaz de obtener los datos necesarios de las imágenes de personas.

- **Registrar Vídeo**: el sistema debe ser capaz de guardar en el servidor un vídeo enviado por el usuario.

3.1.6. Arquitectura

Una vez se conocen los requisitos de la solución y sus restricciones, así como sus casos de uso, se determina su arquitectura. En este caso, se propone un pequeño sistema que cuenta con un cliente, una API Rest y un modelo de Deep Learning ya entrenado.

En la siguiente imagen, se ilustra la arquitectura de la solución propuesta.

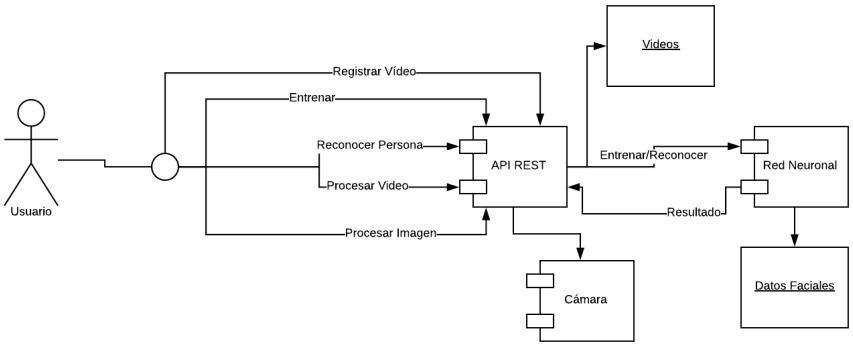

Figura 3.2: Arquitectura propuesta

De izquierda a derecha: un cliente, que en la imagen se ha elegido un teléfono móvil, pero podría ser perfectamente cualquier otro tipo de cliente; una API Rest que hará de intermediario entre el cliente y el modelo de Deep Learning; y, finalmente, la red neuronal utilizada para el reconocimiento facial de personas.

La comunicación entre el cliente y la API Rest será mediante el protocolo HTTP. La API podrá estar conectada a una o varias cámaras, ya sea directamente en el servidor o a través de una cámara IP. También tendrá acceso a un directorio donde estarán almacenados los vídeos que reciba de los diferentes clientes, los cuales podrá procesar; también podrá realizar opera-

ciones como predecir o entrenar sobre la red neuronal. Esta última también tendrá acceso a los datos necesarios para entrenar el sistema.

Cabe destacar que, como la mayoría de herramientas elegidas tienen soporte para Python, este lenguaje de programación es el que ha sido elegido para el desarrollo de todo el proyecto.

3.1.7. Diagramas de secuencia

En esta sección se ilustrarán los diagramas de secuencia para los casos de uso *entrenar*, *reconocer persona* y *procesar vídeo*.

Diagrama de secuencia para entrenar

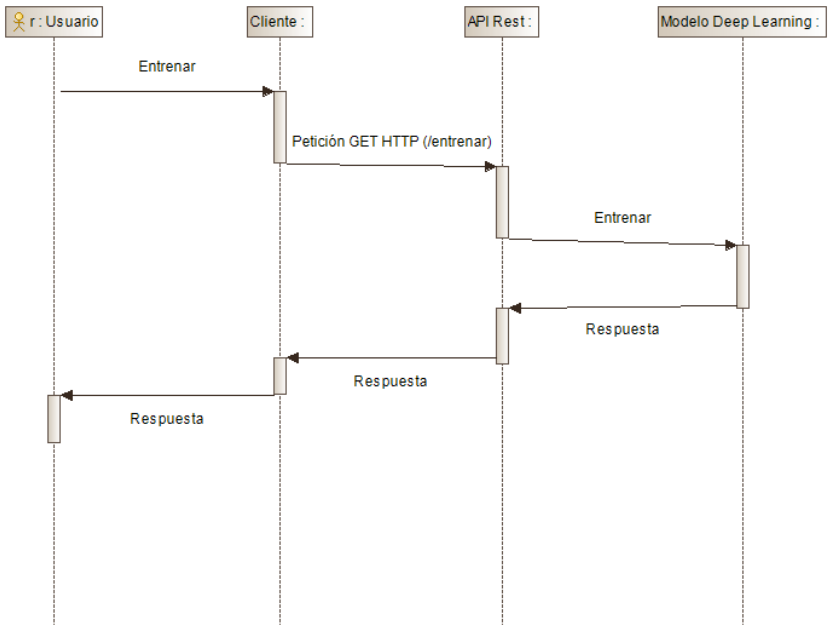

Figura 3.3: Diagrama de secuencia del caso de uso *entrenar*

En el diagrama de secuencia anterior3.3 se ilustra el proceso de entrenamiento del sistema. En primer lugar el usuario realiza la acción que desencadena el proceso de entrenamiento del sistema. Es en este punto cuando el

cliente manda una petición HTTP de tipo GET al *endpoint* /entrenar"de la API Rest, desde donde ejecuta el entrenamiento de la red neuronal.

Diagrama de secuencia para reconocer persona

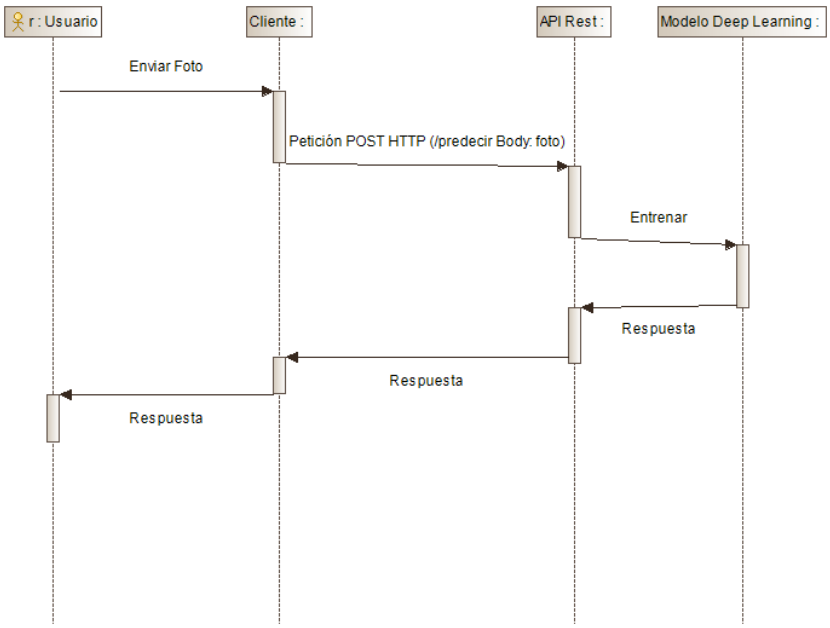

Figura 3.4: Diagrama de secuencia del caso de uso *Reconocer Persona*

En el diagrama de secuencia para reconocer persona 3.4 se ilustra el proceso de predicción en la red neuronal. Para ello el usuario envía una foto a través del cliente a la API mediante una petición HTTP de tipo POST al *endpoint* /predecir. Esta petición almacena en el cuerpo de la misma la fotografía con una cara. La API Rest se comunica con el modelo de Deep Learning y es éste el que devuelve a quién pertenece la cara, o si es desconocido.

Diagrama de secuencia para procesar vídeos

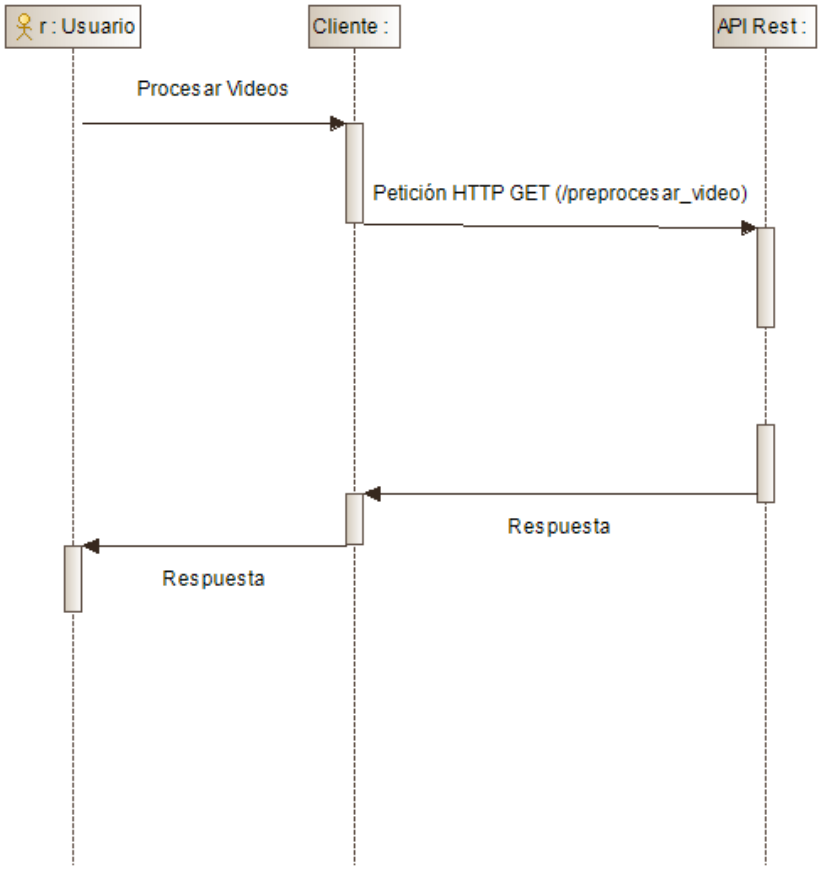

Figura 3.5: Diagrama de secuencia del caso de uso *Procesar Vídeos*

En este diagrama 3.5 se muestra el procedimiento del procesado de vídeos. Es simple, el usuario realiza la acción desencadenante en el cliente y éste manda una petición HTTP al *endpoint* /preprocesar_video". El sistema realiza la acción encomendada y manda una respuesta cuando finaliza el proceso.

3.1.8. Gestión del Proyecto

En este apartado se adjunta una imagen con el diagrama de Gantt que hace referencia a la gestión del proyecto. Desde el aprendizaje de los conceptos

básicos en Machine Learning y Deep Learning hasta las pruebas de validación del sistema desarrollado.

Figura 3.6: Diagrama de Gantt para la gestión del proyecto

En la imagen anterior [3.6] se puede ver cómo se reparte la gestión del proyecto. En primer lugar se realiza un curso de Udacity de una duración de dos semanas aproximadamente que ofrece una formación básica en los conceptos de Machine Learning y, tras la finalización del curso, se realizan diferentes tutoriales sobre Deep Learning. Posteriormente, viene una semana en la que se realizan y se acuerdan los diferentes requisitos que tiene el sistema.

Tras la recolección de requisitos, se establecen dos semanas para el análisis y diseño del sistema a desarrollar. Es de dos semanas debido a que, definidos los casos de uso y diagramas de secuencia del proyecto, se estudian las diferentes alternativas tecnológicas para el desarrollo del sistema propuesto.

El desarrollo del sistema tiene una duración de 4 semanas. Este desarrollo se divide en subtareas: desarrollo del modelo de Deep Learning, desarrollo de la API Rest y del cliente. El proceso más largo es el del desarrollo del modelo de Deep Learning, pues no se está familiarizado con esta tecnología, por lo que pueden surgir diferentes problemas a lo largo del desarrollo de esta parte del sistema.

La validación tecnológica tiene una semana de duración, ya que, al tener todo el sistema desarrollado, se pueden pruebar todos los casos de uso posibles, tal y como se comentará más adelante, en el apartado de *Validación Tecnológica* del presente documento.

3.2. Implementación y desarrollo

En esta sección se detallará todo lo referente al desarrollo de la arquitectura, desde el modelo de Deep Learning, con su entrenamiento incluído; hasta el desarrollo de un pequeño bot de Telegram que hará las veces de cliente, pasando por la API Rest, dejando así toda la arquitectura a desarrollar correctamente documentada.

3.2.1. Desarrollo del modelo de Deep Learning

En esta sección se explicará todo lo relacionado con el desarrollo del modelo de Deep Learning definido para este proyecto, así como el entrenamiento

del mismo y cómo interactuar con el para reconocer personas.

Definición del modelo

En primer lugar, para el desarrollo completo de la arquitectura propuesta anteriormente, es preciso diseñar el modelo de Deep Learning. Como se ha mencionado anteriormente, para simplificar el trabajo se utiliza *una capa* por encima de Tensorflow, que es Keras, por lo tanto, el lenguaje de programación empleado es Python.

Para el desarrollo de este modelo, se define una clase *Modelo* que cuenta con todas las capas de neuronas estimadas para el desarrollo y entrenamiento del mismo. El esquema de la red neuronal puede verse reflejado en la siguiente tabla[3.2.1].

Capa (Tipo)	**Shape**	**Número Parámetros**
dense_1 (Dense)	(None, 4096)	528384
activation_1 (Activation)	(None, 4096)	0
dropout_1 (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 4096)	16781312
activation_2 (Activation)	(None, 4096)	0
dropout_2 (Dropout)	(None, 4096)	0
dense_3 (Dense)	(None, 4)	16388
activation_3 (Activation)	(None, 4)	0

Cuadro 3.1: Estructura de capas de la red neuronal

En base a la tabla anterior, el total de parámetros *entrenables* son 17326084. Otro aspecto a destacar es que, aunque *Activation* y *Dropout* no son capas neuronales realmente, Keras ofrece esta información mediante el método *summary* de la clase *Sequential*, que es la base del modelo.

A continuación se explica cada uno de los tipos que aparecen junto al nombre de cada capa de la tabla anterior.

- **Dense**: Es un tipo de capa donde todas las neuronas que la componen están interconectadas entre ellas.
- **Activation**: Aplica una función de activación a la capa definida previamente para su salida.
- **Dropout**: Aplica *Dropout* a la capa definida anteriormente. El *Dropout* consiste en poner a cero una fracción seleccionada aleatoriamente de los datos de entrada de la capa en cada actualización durante el entrenamiento. Esto ayuda a prevenir el *overfitting*.

Seguidamente se describe cada capa reflejada en la tabla anterior, dando así una visión más detallada del modelo desarrollado, así como los parámetros correspondientes a cada una de ellas, ya que no aparecen reflejados en la tabla.

- **dense_1**: es la capa de entrada de la red y tiene todas sus neuronas interconectadas entre ellas. Cuenta con 4096 neuronas, y las dimensiones de entrada de esta capa es de 128. Es decir, la longitud del vector de datos es de 128.
- **activation_1**: como se ha descrito antes, define la función de activación. En este caso, la función es *relu*. Esta función permite el paso de todos los valores positivos sin cambiarlos, pero asigna todos los valores negativos a 0.[31]
- **dropout_1**: el parámetro de este *dropout* es de 0.5. Esto quiere decir, que la porción de datos que se ponen a 0 se corresponde con el 50 %
- **dense_2**: esta capa es la segunda capa donde están todas las neuronas conectadas entre ellas. Se corresponde con la capa oculta de la red neuronal. Al igual que la de entrada, cuenta con 4096 neuronas.
- **activation_2**: la función de activación para la capa oculta es *relu*. Es la misma función de activación que en la capa de entrada.
- **dropout_2**: al igual que en el caso anterior, se introduce una capa de *dropout* cuyo parámetro es de 0.5, lo que hace que se corresponda con un 50 % de los datos, tal y como ocurre en la primera capa.
- **dense_3**: es la capa de salida de la red neuronal. En la tabla anterior cuenta con 4 neuronas, pues, en el momento de redacción de este apartado se cuenta con cuatro clases para clasificar: tres personas y *desconocido*.

- **activation_3**: la función de activación para la capa de salida de la red neuronal es *softmax*. Esta función de activación se utiliza cuando la clasificación es entre varias clases, como es el caso de este proyecto. *Softmax* asigna probabilidades decimales a cada clase, sumando entre todas 1.[32]

Se ha elegido este número de capas debido a que, tras realizar las pruebas de desarrollo, han dado buenos resultados. El desarrollo comenzó inicialmente se empezó con el doble de neuronas en cada capa y 6 capas en total. Se fue reduciendo el número de capas y neuronas pues, al trabajar sobre un procesador lento inicialmente, se bloqueaba el equipo y se llenaba la memoria del mismo. Tras la reducción de capas y neuronas, esta arquitecturá ofreció resultados óptimos durante el entrenamiento, sin bloquearse.

Una vez definido el modelo de la red, es necesario compilarlo. Para ello, la clase *Sequential* de Keras ofrece un método *compile*, al cual se le pasan varios argumentos que sirven como parámetros para, posteriormente, entrenar y utilizar correctamente el modelo de Deep Learning definido. Los parámetros que se le especifican a Keras para que compile el modelo son los siguientes:

- ***loss***: este parámetro es obligatorio y define la función de pérdida o función de optimización de la puntuación. La función que se la asigna es *categorical_crossentropy*, que mide el número medio de bits necesitados para identificar un evento en un conjunto de probabilidades. Se usa para clasificar entre distintas clases.[33]
- ***optimizer***: es la función de optimización. Los algoritmos de optimización ayudan a minimizar o maximizar una función objetivo, que es una pequeña función matemática dependiente de los parámetros de aprendizaje del modelo.[34]. Este parámetro es también obligatorio, y se ha seleccionado como optimizador *RMSProp*, que divide la tasa de aprendizaje por un peso por un promedio continuo de la magnitud de gradientes recientes para ese peso.[35] El ratio de aprendizaje que se le ha especificado es el indicado por defecto, que es del 0,1 %.
- ***metrics***: este otro parámetro es utilizado para indicar cuáles son las métricas que se usarán para evaluar el modelo definido. En el caso de este proyecto simplemente se utiliza *accuracy*, que se refiere a la precisión con la que clasifica a una persona, es decir, la precisión con la que la reconoce.

Una vez definido y explicado el modelo expuesto anteriormente, el si-

guiente paso es entrenarlo y cómo se realiza este paso. La explicación de este proceso y todo lo relacionado con él se realiza en el siguiente apartado.

Entrenamiento del modelo

Para que el modelo sea capaz de realizar predicciones con mayor o menor precisión para dar un resultado aceptable en el reconocimiento facial es necesario entrenar la red neuronal. En este apartado se explica todo el proceso y cómo se ha entrenado.

Cabe destacar que para el proceso de entrenamiento, Keras cuenta con una función *train* en la clase *Sequential*, lo que hace que el proceso de entrenamiento sea mucho más sencillo y transparente al desarrollador.

Antes de comenzar el entrenamiento de la red neuronal con los vectores de características de las caras obtenidos mediante *Dlib*, cuyo proceso se explicará más adelante, se define un *checkpointer*. Los *checkpointers* guardan los pesos actuales si son mejores que los anteriores. Esto es útil cuando, por ejemplo, se aborta el proceso de entrenamiento por algún tipo de error. Este objeto *ModelCheckpoint* de la librería de Keras tiene tres parámetros:

- ***filepath***: ruta donde se van a guardar los pesos obtenidos. En el caso de este proyecto se guardan en *datos_red/pesos.hdf5*.
- ***verbose***: no es necesario, pero al tenerlo a 1, muestra un mensaje por pantalla informando de cuando ha establecido el sistema un punto de control con los pesos.
- ***save_best_only***: con *True* solamente guarda el mejor hasta ahora. Si en algún momento baja la precisión, esos pesos no los guarda.

Una vez definido el *checkpointer*, es necesario preprocesar los datos que se van a utilizar para entrenar y validar el modelo de Deep Learning definido en el apartado anterior.

El preprocesado se hace a partir de los datos obtenidos de *Dlib*. Estos datos se almacenan en directorios. La estructura de estos directorios es *PersonasAutorizadas* y dentro de esta carpeta se encuentran el resto de directorios

con el nombre de la persona autorizada y los datos en formato *pkl* en su interior. Se abre el fichero serializado y se toma como etiqueta de los datos el nombre de la persona que está en el directorio. Esta acción se repite para cada uno de los directorios dentro de *PersonasAutorizadas*.

Cada vez que se termina de procesar los datos de una persona, se dividen en dos conjuntos, destinados a entrenar y a validar. El tamaño del conjunto de tests es de un 33 % del total, con una *semilla* aleatoria de 42. Esta acción se realiza para cada iteración del bucle donde se procesan los datos faciales de cada persona.

Cuando se termina de procesar los datos de las personas autorizadas, es el turno de las personas que son desconocidas. Para esta acción se realiza exactamente lo mismo, solo que el fichero serializado *pkl* se encuentra en un directorio *Desconocido* situado en la raíz del proyecto.

También se vuelcan las etiquetas utilizadas a un fichero serializado *pkl* para poder acceder a ellas posteriormente de forma rápida en lugar de listar el directorio y obtener los nombres de las personas y *Desconocido* para mantenerlos en memoria.

Justo antes de comenzar el entrenamiento se procesan las etiquetas, convirtiéndolas a *1, 2, 3...*. El proceso de entrenamiento se realiza con el método *fit* que ofrece Keras. Los parámetros para este método son los siguientes:

- ***Datos de entrenamiento***: son los datos obtenidos anteriormente para entrenar.
- ***Etiquetas de entrenamiento***: son las etiquetas que se corresponden con los datos anteriores.
- ***batch_size***: se corresponde con la cantidad de datos que se le enseñan a la misma vez a la red neuronal. Está definido actualmente en 150, que se justificará más adelante en la sección de pruebas.
- ***nb_epoch***: número de veces que se repite la operación de entrenamiento, es decir, las veces que se itera sobre el conjunto de datos.
- ***validation_data***: es una tupla donde se especifican los datos utilizados para validación, que se definieron anteriormente como *test*.
- ***callbacks***: un vector donde se especifican los callbacks[36] de Keras. En este caso se introduce el *ModelCheckpointer* definido anteriormente.

- ***shuffle***: a True especifica que los datos se mezclan en cada iteración sobre el conjunto de datos, cambiando así el orden de los mismos y ayudando a reducir el *overfitting*.

3.2.2. Obtención de los datos faciales

La obtención de los datos de características faciales para el entrenamiento de la red se puede hacer por tres vías: mediante una cámara, a través de un vídeo o por imágenes. En este apartado se describirá la forma en la que se realiza cada una de ellas, aunque la utilizada en el cliente a desarrollar –que se verá posteriormente– es a través de vídeo, aunque para cada una de estas tres vías existe un *script* que realiza las funciones.

Las tres vías anteriores comparten, prácticamente, el mismo mecanismo: ya sea una fotografía, o un frame de la cámara o del vídeo, el modelo preentrenado de *Dlib* detecta algún rostro en la entrada. En caso de que detecte el rostro, el propio modelo obtiene, en un eje de coordenadas (x, y), las posiciones de distintos aspectos faciales pertenecientes a la persona que se esté procesando. Estos datos son los que se almacenan para, posteriormente, alimentar a la red neuronal durante el entrenamiento. El mismo mecanismo es usado para el reconocimiento de personas una vez entrenado el sistema.

El modelo preentrenado de *Dlib* reconoce 68 puntos faciales, entre los que están la nariz, las cejas, la boca y la forma de la cara. Esto simplifica mucho el trabajo a la hora de obtener los vectores de características faciales. En la siguiente imagen[3.7] puede observarse de forma visual cuáles son estos puntos que es capaz de reconocer el modelo preentrenado.

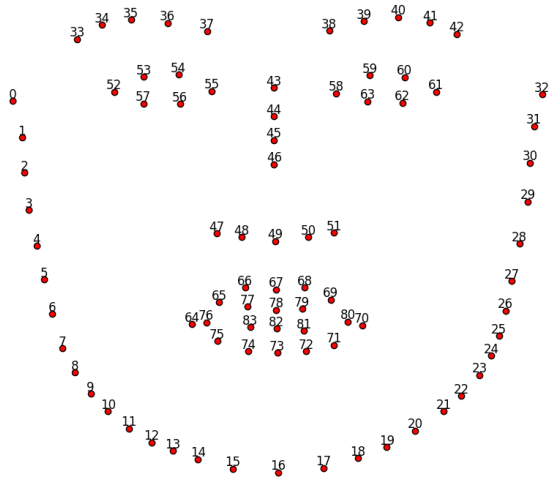

Figura 3.7: Puntos faciales reconocidos por Dlib

Recopilación de características a través de una cámara

Para la correcta recopilación de los vectores de características faciales a través de una cámara es estrictamente necesario contar con una conectada al equipo donde se ejecutará esta función, o bien contar con una cámara IP accesible desde el equipo, ambas con buena calidad.

Por defecto, el *script* implementado para esta función toma la primera cámara conectada al ordenador. La ejecución de este *script* se realiza a través de la consola con el siguiente comando:

```
$ python3 capturar_camara.py
```

Antes de comenzar a capturar frames de la cámara se solicitará el nombre de la persona a registrar, creando así un directorio en *PersonasAutorizadas* con dicho nombre, donde se almacenará un fichero serializado *pkl* con los vectores de características de dicha persona. En caso de que exista el fichero, simplemente se añaden los nuevos datos que se recogerán posteriormente a los ya existentes.

Tras la creación del directorio, se abre el flujo de la cámara – si no se

pudiera, lanzaría un error– y una ventana de OpenCV mostrando el flujo. En cuanto detecta una cara dibuja un cuadrado verde a su alrededor, gracias a *Dlib*.

Para comenzar a capturar datos, se pulsa la tecla *r* y cambia de color el cuadrado a rojo. El tiempo medio óptimo para la captura de datos es de unos 30-45 segundos, que se justificará posteriormente. Para parar de obtener datos se vuelve a pulsar *r* o *q* para finalizar directamente la ejecución del *script*.

Durante el proceso de captura de datos faciales para entrenamiento mediante un modelo preentrenado de *Dlib* destinado a este fin, visto al inicio de esta sección del documento, se van añadiendo los diferentes vectores obtenidos de cada frame durante los 30-45 segundos y al finalizar el proceso de obtención de los datos, se vuelcan al fichero *face_descriptor.pkl*.

Recopilación de características a través de un vídeo

La recopilación de características se realiza prácticamente de la misma forma que en el apartado anterior. La diferencia radica en que, en lugar de obtener los frames a través del flujo de la cámara, se hace a través de un vídeo almacenado, en formato *mp4*. Como en el caso anterior, con un vídeo de una duración de unos 30-45 segundos sería suficiente.

La ejecución del *script* desarrollado que se encarga de obtener los datos faciales, mediante Dlib, correspondientes a la persona que aparece en el vídeo se realiza de la siguiente forma desde consola:

```
$ python3 capturar_video.py -p ruta -n nombre
```

Los parámetros necesarios, ambos obligatorios, para la ejecución del *script* son los siguientes:

- ***-p / --path***: define la ruta donde está situado el vídeo.
- ***-n / --name***: indica el nombre de la persona a la que le corresponde el vídeo.

Una vez se comience el procesado del vídeo, lo primero que hace el *script* es comprobar que la extensión del vídeo es *mp4*, que en caso contrario muestra un mensaje de error por pantalla y finaliza el proceso. Si la extensión del archivo es *mp4*, se comprueba si no existe un directorio en *PersonasAutorizadas* con el nombre de la persona a la que le corresponde el vídeo, creándolo en este caso u obteniendo los datos ya almacenados en caso de que exista para añadirle los nuevos datos.

Mientras el vídeo no se haya acabado se procesa frame a frame como en el caso anterior, detectando la cara – en el caso de que el frame sea correcto– y obteniendo los vectores de características a partir de los 68 puntos faciales que define *Dlib*.

Cuando el proceso ha terminado, vuelca los datos a al fichero *face_descriptor.pkl* situado en el directorio correspondiente a la persona y muestra un mensaje informando de la finalización del proceso.

Recopilación de características a través de imágenes

Como se ha mencionado al principio de la sección de este documento, la obtención de características faciales a través de imágenes es la última de las tres formas de obtener datos para entrenar el modelo a partir de los rostros humanos.

Cabe destacar que este método de obtención de datos es el que mayor cantidad de datos de entrada necesita, es decir, necesita un volumen de fotografías faciales muy elevado, por ello, es el único que no se utiliza posteriormente en la arquitectura.

La ejecución del *script* que obtiene los datos faciales correspondientes a la persona que aparece en las fotografías se realiza de la siguiente forma desde consola:

```
$ python3 capturar_imagen.py -p ruta
```

El parámetro *-p* es opcional e indica la ruta hacia el directorio donde están las imágenes. Si este parámetro no se indica, se tomará por defecto el directorio *imágenes* situado en el mismo sitio que el *script*. La estructura

de directorios donde se almacenan las imágenes debe estar compuesta de la siguiente forma:

Figura 3.8: Estructura de directorios de la carpeta imágenes

En la imagen anterior 3.8 se puede ver que las imágenes pertenecientes a cada persona se almacenan en una carpeta con su nombre. Esto último es algo muy importante, pues el nombre de la persona se tomará a partir de ahí. El nombre de la persona se corresponde con la clase que utiliza la red neuronal para dar una predicción, es decir, la etiqueta que le pone a los datos de entrada.

El proceso de obtención de datos es bastante simple. En primer lugar se lista cada uno de los directorios, tomando como nombre de la persona el nombre del directorio. Posteriormente se procesa imagen a imagen, detectando la cara de la persona y obteniendo los vectores de características de cada una de las fotos. Al finalizar el proceso, se creará un fichero *pkl* dentro de un directorio con el nombre de la persona en *PersonasAutorizadas*. Cabe destacar que, para evitar que se intenten procesar ficheros que no son imágenes, se añade una restricción a formatos *jpg*, *gif* y *png*.

Posteriormente se verá que, debido al cliente que se implementa, esta funcionalidad no se ofrece a través de la API, ya que, como se ha mencionado, requiere de una gran cantidad de fotografías para que el sistema pueda ser entrenado correctamente.

Entrenamiento del modelo

Para que el modelo desarrollado pueda ser entrenado se ha desarrollado también un *script* para este propósito. Este *script* se entrega junto a los ante-

riores y tiene el nombre de *entrenar_modelo.py*. Para proceder al entrenamiento de la red neuronal descrita en este apartado, se ejecuta desde la terminal del sistema operativo el siguiente comando:

```
$ python3 entrenar_modelo.py
```

Cuando el proceso finalice, guardará los mejores pesos encontrados para poder reconocer posteriormente personas. Cabe destacar que este *script* debe estar situado tal y como se entrega en el proyecto, es decir, en el mismo lugar donde se encuentra el paquete *Modelo* y la carpeta *datos_red*, así como las carpetas *PersonasAutorizadas* y *Desconocido*.

Reconocimiento facial a través de una cámara

En esta sección se describe el uso de un *script* para reconocimiento facial a través de una cámara, partiendo del modelo ya entrenado. Este *script* ha sido implementado como prueba de que, evidentemente, el sistema funciona correctamente y no da ningún tipo de error.

Para poder ejecutar este *script*, en primer lugar es necesario tener una cámara conectada al equipo, ya sea de forma local o vía IP. Disponiendo de una cámara conectada al equipo, se puede proceder a la ejecución del *script*. Desde la línea de comandos se puede ejecutar con el siguiente comando:

```
$ python3 reconocimiento_cara.py
```

El programa trata de abrir el flujo de la cámara y, una vez abierto, mostrará una ventana de OpenCV que muestra los *frames* recogidos por la cámara. En cada frame, como en el caso del procesado de datos desde vídeo, se trata de detectar algún rostro. Con el rostro detectado, obtiene los vectores de características con *Dlib* y se los transfiere a la red neuronal mediante el método *predict*. La salida de esta función es la etiqueta que corresponde a la predicción, la cual se utiliza para mostrar el nombre de la persona reconocida en la ventana de OpenCV, por encima del cuadrado, con la probabilidad con la que se ha predicho.

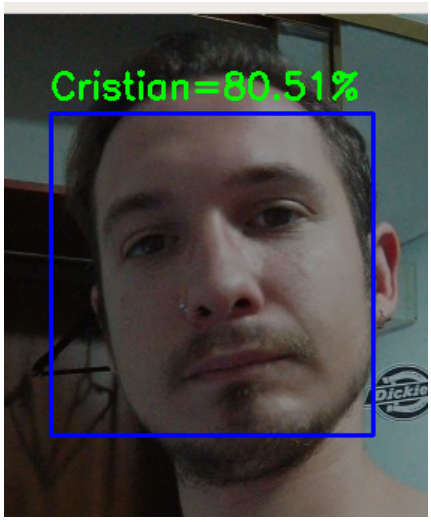

Figura 3.9: Reconocimiento facial

Para terminar la ejecución del programa basta con pulsar la tecla *q*. Cabe destacar que, si se desea conectar con una cámara IP, basta con pasar la cadena de conexión a la misma como parámetro del método *cv2.VideoCapture*, que se puede encontrar en el código fuente del *script*, en la línea 29.

El umbral de probabilidad está en el 80 %. Por debajo de este umbral no se mostrará ningún tipo de etiqueta reconocida, ya que, en ocasiones, el detector facial de *Dlib* arroja falsos positivos y encuentra rostros donde no los hay, evitándose así que se muestren etiquetas completamente erróneas. El por qué de este valor se justificará posteriormente en el apartado de pruebas.

3.2.3. Desarrollo de la API Rest

Como se comentó en el apartado de las tecnologías utilizadas, el *framework* elegido para el desarrollo de la API Rest de este proyecto es Flask[26]. Este *framework* ofrece una gran facilidad a la hora de definir *endpoints* y los métodos asociados a los mismos. En este proyecto se han definido varios *endpoints* que se explican a continuación:

- ***"predecir"***: solamente acepta peticiones HTTP de tipo POST. En el cuerpo de la petición se le manda una imagen en *base64*. Una vez se

recupera el fichero en esta codificación, se decodifica y se obtienen los vectores de características faciales de la persona del a imagen tal y como se ha visto anteriormente. Obtenidos los datos necesarios, se le pasan al método *predict* que ofrece Keras, devolviendo una etiqueta que se utiliza para obtener el nombre de la persona. El umbral de probabilidad para la predicción es también del 80 %. La respuesta de este método de la API es un objeto JSON con las personas reconocidas en la imagen.

- ***"video"***: al igual que el *endpoint* anterior, solamente acepta peticiones HTTP de tipo POST. En este caso, la funcionalidad es almacenar un vídeo en el servidor. Este vídeo se envía en el cuerpo de la petición, siendo éste de tipo *multipart/form-data* del que se recupera el vídeo completo. El vídeo se guardará en una carpeta llamada *vídeos* y el método devolverá una cadena de texto informando de que ha guardado correctamente el vídeo o informando de lo contrario en caso de que haya algún error. El nombre con el que se guarda el fichero de vídeo se corresponde con el de la persona a la que pertenece.

- ***"preprocesar_video"***: este método acepta peticiones tipo POST. El cuerpo de la petición no lleva ningún tipo de datos. Este método se encarga de recorrer todos los ficheros que se encuentran en el directorio *vídeos* y obteniendo los datos necesarios para el entrenamiento mediante el método visto anteriormente. El nombre de la persona a la que corresponde el vídeo es el mismo que el nombre del archivo.

- ***"entrenar"***: este *endpoint* realiza lo que su propio nombre indica: entrena la red neuronal con los datos almacenados. La funcionalidad de este *endpoint* simplemente es llamar a la función de entrenamiento del modelo.

Durante el desarrollo y las pruebas de la API se ha utilizado el servidor *embebido* con el que cuenta el *framework* utilizado, en modo *debug* para que aparezca mayor información sobre los errores que se pudieran ocurrir. El puerto en el que escucha es el 5000, aunque durante la fase de despliegue puede cambiar debido a diferentes factores.

3.2.4. Desarrollo del cliente

Como cliente se ha elegido desarrollar un pequeño *bot* de la aplicación Telegram. El motivo por el que se desarrolla este software es por su escasa

complejidad y que, además, sirve para ilustrar perfectamente el funcionamiento de la arquitectura propuesta con anterioridad 3.2.

El bot ha sido desarrollado utilizando el lenguaje de programación Python, para mayor homogeneidad con el sistema. La librería utilizada en el desarrollo de este bot es *Python-telegram-bot*, que se puede encontrar en Github.[37]. Este bot se puede encontrar en Telegram con el nombre de *@tfg-cristianbot*.

Antes del comienzo del desarrollo, se ha registrado un *token* con el *bot father* para poder hacer uso de la API de Telegram y poder desarrollar el pequeño software. Como la gran mayoría de los bots de Telegram, este también cuenta con una serie de comandos que se inician con / y se escribe el nombre del mismo detrás. Cabe destacar que este bot hace uso de la API Rest definida en el apartado anterior. A continuación se explican los comandos que utiliza el bot.

- ***"/registrar_video"***: este método es, como su propio nombre indica, para registrar un vídeo en la API Rest anterior mediante el *endpoint* definido como *"/video"*. Su funcionamiento es bastante simple: se manda el comando y el bot contestará con un mensaje solicitando el nombre de la persona de la que se va a pasar el vídeo. Hecho esto, dirá que se mande un vídeo de la cara de unos 30-45 segundos desde diferentes ángulos para guardarlo en el servidor. Cabe destacar que el usuario solamente podrá tener un vídeo almacenado en el servidor, es decir, si el mismo usuario reenvía un nuevo vídeo, el anterior se sobreescribe.
- ***"/abrir_puerta"***: es una pequeña simulación que toma una fotografía de una cámara IP y se la pasa a la API al *endpoint* definido como *"/predecir"*. Una vez conteste la API Rest se determinará si es *Desconocido*. En caso de que la etiqueta obtenida es diferente de *Desconocido*, el bot enviará un mensaje al usuario diciendo que abre la puerta y que el usuario no está autorizado en caso de que sea desconocido.
- ***"/preprocesar_video"***: este comando se comunica con la API para procesar los vídeos almacenados en el servidor de la misma forma que se describió anteriormente. Para agregar seguridad al sistema y evitar que dos usuarios procesen los vídeos a la misma vez, ya que los de Telegram son procesos individuales por cada usuario, se añade un *semáforo* al método que se encarga de atender este comando. De esta forma, los usuarios quedan a la espera de que termine el proceso. Además, para

aún más seguridad se ha definido una lista de administradores del bot y un *decorador*[38] en Python para asegurar que solamente los usuarios de Telegram en esa lista puedan realizar esta acción.

- ***"/entrenar"***: como su propio nombre indica, sirve para entrenar la red neuronal a través de la API, tal y como se describió en el apartado anterior. Cuenta con la misma seguridad, tanto en el acceso concurrente como en el acceso solamente para usuarios autorizados mediante el *decorador* de Python.

Cabe decir que también se le puede enviar una fotografía al bot, sin especificar ningún comando, y éste lo enviará a la API Rest, devolviendo un vector con las etiquetas para las personas detectadas en la imagen y el número de personas que hay en ella.

Capítulo 4

Validación tecnológica

En esta sección del documento se detallarán las pruebas realizadas para comprobar que el sistema desarrollado funciona correctamente. También se incluirá un pequeño manual para el despliegue del sistema, incluyendo la instalación de las librerías necesarias para que se pueda ejecutar todo correctamente.

4.1. Pruebas del modelo de Deep Learning

Antes de comenzar a explicar las pruebas realizadas es preciso describir el entorno con el que se han realizado dichas pruebas, pues existen muchos factores que condicionan la ejecución de las pruebas. Las condiciones del entorno, así como la cámara utilizada durante el desarrollo son las las siguientes:

- Cámara Deportiva Victure AC80 conectada como *webcam* al ordenador.
- Condiciones de luz natural durante el día y, cuando baja la luz solar, luz blanca de tubos fluorescentes.

En primer lugar se enunciarán los motivos de por qué se toma un 80 % de probabilidad y por qué la duración de los vídeos para procesar es de una duración de entre 30 y 45 segundos. El principal motivo por lo que el umbral

se ha establecido en un 80 % de probabilidad es porque, durante el desarrollo y las pruebas que se han ido realizando,probando con diferentes umbrales, el programa en ocasiones arrojaba falsos positivos, se fue subiendo el umbral mínimo hasta llegar al 80 %, que da unos resultados óptimos. La duración de los vídeos se ha establecido entre 30 y 45 segundos porque:

- No todos los frames de un vídeo son válidos, por lo que se descartan algunos de ellos. Un frame no válido es un frame que no se puede leer. OpenCV, en la función *read* devuelve un valor *True* o *False* para indicar si este frame es válido o no.
- Se han realizado pruebas con vídeos de menos duración, dando errores en las predicciones. Se considera un error una predicción que se corresponde con un falso positivo, es decir, etiqueta a una persona con el nombre de otra que no es.
- Con 4 categorías, tras probar diferentes duraciones, entre 30 y 45 segundos suele haber datos suficientes. Estos datos son los vectores de características faciales obtenidos con la librería mencionada anteriormente de *Dlib*. El peso del fichero serializado tras captar un buen número de vectores de características está en torno a los 400KB. En ocasiones el fichero puede pesar algo menos, lo que es indicio de que no todos los frames del vídeo han sido válidos.

Respecto al punto anterior, los vectores de características se obtienen, como ya se ha indicado, a través de Dlib. Esta herramienta es capaz de sacar un vector de características por frame o imagen. Aunque puede parecer poco, este vector de características contiene una longitud de información de 128 elementos, coincidiendo con el tamaño del vector de entrada de la red neuronal descrita anteriormente. Como ya se ha comentado anteriormente, se ha probado con diferentes duraciones de los vídeos a procesar para el entrenamiento del sistema. Se ha considerado, para este caso con 4 categorías, que el número de datos óptimos para que exista un buen entrenamiento y la precisión –que se verá más adelante– del sistema sea adecuada, debe ser de 326 vectores, de 128 elementos cada uno. En total son 41728 datos diferentes correspondientes a las características facial de cada una de las personas.

Las pruebas realizadas con la cámara deportiva se han lelvado a cabo en el entorno definido anteriormente. Las pruebas se realizan con las 4 personas utilizadas durante el entrenamiento del modelo de Deep Learning. Existen

diferentes medidas en validación, pero la mayoría son para clasificación binaria, por ello, para este proyecto se utiliza simplemente la precisión de las predicciones:

- ***Precision***[39]: mide la proporción de identificaciones que han sido correctas. La fórmula matemática que define a esta medida muy utilizada en Machine Learning y Deep Learning es:

$$Precision = \frac{Positivos}{Totaldedetecciones}$$

Teniendo en cuenta la ecuación anterior, se evalúa la eficiencia del modelo desarrollado en el apartado anterior. Keras cuenta con una ventaja que algunos otros *frameworks* no tienen: la capacidad de obtener la precisión del modelo durante el entrenamiento del mismo, a partir de un parámetro llamado *metrics* de la función de entrenamiento. La función de entrenamiento, al finalizar, arroja una precisión del 99 %. Es muy efectivo, y se puede pensar que existe *overfitting*. Esto no es así, pues tiene una explicación lógica: la red cuenta con 4 clases o personas con las que se ha entrenado, por lo que esta red aprende rápido. Es lógico pensar que al aumentar el número de clases a clasificar, este porcentaje de precisión se verá reducido.

Esto se puede demostrar, pues, a lo largo de diferentes días, en diferentes condiciones de luz, se ha probado el modelo a través de la cámara deportiva mencionada anteriormente, con un resultado satisfactorio. Como por ejemplo, en uno de los casos, se ha situado a 4 personas delante de la cámara y han sido reconocidas perfectamente por la red neuronal. Existen algunas situaciones en las que arroja algún falso positivo, pero sobre todo ocurren a partir de la cámara de un Samsung Galaxy S7 Edge, enviando una fotografía desde el cliente desarrollado.

Respecto a la función de entrenamiento, se ha establecido el número de *epochs* en 50, es decir, se va a iterar 50 veces sobre el conjunto de datos de entrenamiento; y el tamaño del lote – *batch_size*– en 150. Se ha ido probando de 5 en 5 epochs, partiendo desde 5. 50 son más que suficientes, pues en torno al *epoch* número 25, con 4 categorías, no mejora demasiado la red. Se planea en 50, por si en algún momento se decidiera utilizar un mayor número de categorías, pues le costaría más aprender a la red neuronal.

4.2. Pruebas de la API Rest

Las pruebas realizadas sobre la API Rest han sido bastante simples. Se han probado los diferentes *endpoints* definidos con Postman[40], una aplicación para mandar peticiones HTTP a servicios web. Se han probado las siguientes situaciones:

- Se envía un vídeo al *endpoint "/video"* esperando que se devuelva la cadena de texto *Guardado*.
- Se envía una foto de una persona conocida al *endpoint /predecir* esperando que la API responda con el nombre de la persona, tras llamar al método *predecir* de la red neuronal.
- Se envía una fotografía de una persona desconocida al *endpoint "/predecir"* esperando que se devuelva la cadena *Desconocido*.
- Se envía una petición POST al método *"preprocesar_video"* y se comprueba que se ha realizado la acción correspondiente.
- Se envía una petición POST al *endpoint "/entrenar"* y se espera que se acabe el proceso, momento en el que se enviará una respuesta informando de la finalización del entrenamiento. Se comprueba en el *log* que, efectivamente, se está entrenando el modelo.

Realizando las acciones definidas en la lista anterior con Postman, se comprueba que la API Rest definida previamente en el apartado de desarrollo, así como todos sus *endpoints* funciona como es esperado. Sabiendo que la API Rest funciona correctamente, se puede proceder a las pruebas del bot de Telegram desarrollado como cliente.

4.3. Pruebas del cliente desarrollado

Al igual que las pruebas realizadas sobre la API Rest, las que se han realizado sobre este bot de Telegram son muy básicas, pues este bot no ofrece más que los comandos definidos previamente y la posibilidad de recibir una foto y que mande un mensaje al usuario diciendo quiénes hay en la foto y el número de personas detectadas. Las pruebas realizadas son:

- Se envía una foto y se espera la respuesta por parte de la API Rest con los nombres y el número de personas.
- Se manda el comando *"registrar_video"*, donde solicita el nombre y el vídeo. Se manda y se espera una respuesta de la API como que el vídeo ha sido recibido.
- Se ejecuta el comando *"abrir_puerta"*. Se espera la respuesta por parte de la API.
- Se ejecutan los comandos *"entrenar"* y *"preprocesar_video"* desde un teléfono cuyo usuario no tiene permisos de usuario, esperando el mensaje de que no se está autorizado.
- Se ejecutan los comandos del punto anterior desde un usuario autorizado y se comprueba que, como en el caso de la API Rest, funciona correctamente.
- Para controlar el acceso concurrente, se ejecuta el comando *"preprocesar_video"* desde dos usuarios diferentes que se encuentran en la lista de administradores. Se comprueba que, efectivamente, el primero en llegar adquiere el semáforo y el segundo queda bloqueado a la espera de que termine el proceso.

Comprobadas todas las situaciones anteriores, queda demostrado que el cliente funciona correctamente. Las pruebas son bastantes simples, pues este bot solamente hace de intermediario, ya que todo el trabajo realmente lo realiza la API Rest definida anteriormente.

Capítulo 5

Conclusiones

En este apartado se exponen las conclusiones obtenidos a partir del desarrollo del sistema descrito durante todo el documento. Se dividen en dos partes, tanto personales como del proyecto en sí.

5.1. Conclusiones del proyecto

Tras finalizar el desarrollo y el presente documento, se han llegado a varias consideraciones. En primer lugar, comentar que, a pesar de ser un proyecto tedioso, porque requiere mucha *prueba* y *error* durante el proceso de validación tecnológica de la solución desarrollada y requiere mucho tiempo y trabajo; cumple con los objetivos expuestos anteriormente y que, además, es bastante sencillo de editar, ya que al estar escrito en Python, muchas cosas son triviales.

Para continuar con este apartado, cabe destacar que el sistema requiere de bastante potencia para su entrenamiento, debido a al tiempo que toma si se usa sobre un procesador lento, como ocurrió el principio del desarrollo del proyecto, durante la investigación de las tecnologías y el aprendizaje de Keras.

Por último, decir que este sistema no es completamente perfecto, cuenta con taras que quizá se podrían subsanar con aportaciones económicas y con un equipo de desarrollo más grande. Por esto mismo, este sistema se puede mejorar, por ejemplo, en cuestión de seguridad. En la siguiente tabla puede

observarse ver una relación entre objetivos conseguidos y objetivos no conseguidos:

Objetivo	**¿Conseguido?**
Detección Facial	Si
Reconocimiento Facial	Si
Personas Desconocidas	Si
Utiliza Redes Neuronales	Si
Control de acceso concurrente desde cliente	Si
Comunicación HTTPS	No
Abre puerta	No
Sistema de Login API	No

Cuadro 5.1: Relación de objetivos conseguidos y no conseguidos

5.2. Conclusiones personales

Como conclusiones personales solo puedo decir que este proyecto me ha hecho investigar mucho sobre un tema que me parece tan impresionante como es el Machine Learning. Gracias a él he aprendido Python y distintos conceptos de Machine Learning y Deep Learning, aunque sean conceptos básicos.

También creo que he mejorado en las labores de investigación, ya que, al tener que diseñar una arquitectura más o menos compleja, he tenido que valorar diferentes alternativas tecnológicas en distintos lenguajes y elegir el que más adecuado me ha parecido y, además, solventar los diferentes problemas que se han ido presentando a lo largo del desarrollo, que no han sido pocos.

En definitiva, creo que este Trabajo de Fin de Grado me ha aportado mucho tanto en lo personal como en lo profesional.

Capítulo 6

Trabajos Futuros

En esta sección del documento se enuncian los posibles trabajos futuros, ya que este sistema tiene algunas carencias que no se han subsanado debido a que escapaban del ámbito de este Trabajo de Fin de Grado.

6.1. Sistema de *login* para la API Rest

La API Rest está completamente accesible desde cualquier punto siempre y cuando se conozca la dirección de la misma, por lo que cualquiera podría subir un vídeo y reentrenar el sistema o utilizar la API Rest desarrollada de forma deliberada.

Esto se podría subsanar mediante un sistema de *login* o autenticación como podría ser el famoso JWT o Json Web Token. Este sistema lo que hace es crear un *token* a partir de, por ejemplo, un usuario y su contraseña, junto a una clave secreta. A este *token* se le otorga una duración, que al caducar es necesario obtener otro. De esta forma, se puede hacer que los *endpoints* deseados requieran de un *token* que debe ser enviado en la cabecera de la petición HTTP para poder ejecutarlo correctamente.

6.2. HTTPS para la API Rest

Para que el sistema fuera aún más seguro, se podría implementar HTTPS para la API Rest, ya que de esta forma, los paquetes que envía el cliente estarían utilizando el protocolo HTTPS donde éstos no se enviarían en texto plano, haciendo la comunicación bastante más segura, sobre todo si se envían datos sensibles como una contraseña para el logueo contra la API y obtener el JWT descrito en el apartado anterio.

6.3. Conectar el bot a una cerradura

El bot de Telegram utilizado a modo de ejemplo que se ha descrito en la sección sobre el desarrollo del sistema tiene un comando llamado */abrir_puerta* que toma la imagen de una cámara y comprueba si es una persona conocida o no. Un trabajo futuro sería conectar este bot con una cerradura para que abra una puerta cuando se ejecute este comando en caso de que sea una persona reconocida y registrada dentro del sistema.

Anexos

Anexos A

Anexo I: Manual de despliegue

En este anexo se explica cómo desplegar el sistema que ha sido desarrollado para este Trabajo de Fin de Grado. Se explica la instalación de librerías necesarias, así como del servidor necesario para desplegar la API Rest en Flask para que se pueda ejecutar correctamente. También se tiene en cuenta que el equipo donde se va a desplegar el sistema cuenta con CUDA instalado. En primer lugar, se explica la instalación de las librerías y herramientas necesarias.

A.1. Instalación de recursos necesarios

Los recursos y librerías necesarias para poder desplegar correctamente el proyecto se pueden instalar desde *pip*, el gestor de dependencias para Python. En la siguiente lista se especifican los comandos que necesitan ejecutarse en una terminal para instalar las librerías y dependencias necesarias. Cabe destacar que se da por supuesto que se tiene CUDA instalado y se usa un equipo Linux, pues es donde se han realizado las diferentes pruebas. La versión de Python utilizada es la 3.6.

- pip3 install numpy
- pip3 install tensorflow-gpu (o sin -gpu en caso de no tener CUDA).
- pip3 install keras

- pip3 install h5py
- pip3 install scikit-learn
- pip3 install flask
- pip3 install opencv-python
- pip3 install python-telegram-bot
- pip3 install requests
- pip3 install gunicorn

Todas estas dependencias se pueden instalar de una sola vez ejecutando *pip3 install -r requirements.txt*. *requirements.txt* es un fichero donde está el nombre de todas las librerías de la lista anterior para instalar todo de una sola vez. Este fichero se encuentra en el repositorio de Github mencionado en la introducción de este documento y en el disco entregado junto a esta documentación. Si todo se ha instalado correctamente, ya se puede empezar a desplegar todo el proyecto.

A.2. Despliegue de la API Rest y el modelo

Una cosa muy importante a tener en cuenta es que el modelo de Deep Learning desarrollado y la API Rest deben estar juntos, para que se puedan servir los métodos del modelo de Keras a través de la API. Como el servidor utilizado para pruebas es el de *debug* que trae embebido Flask y puede dar fallos en producción, se utiliza *Gunicorn* desde la línea de comandos, situados en el directorio donde se encuentra el archivo en Python. El comando para ejecutar el servidor con la API Rest es el siguiente:

```
$ gunicorn –w 4 –b 127.0.0.1:5000 api:app
```

Los parámetros que acompañan al comando se explican a continuación:

- ***-w***: indica el número de *workers* que tendrá. Esto quiere decir el número de peticiones que podrá atender simultáneamente el servidor. En este caso son 4.

- ***-b***: indica la dirección a *bindear*, es decir, la dirección a través de la cual será accesible y el puerto. En este caso se utiliza el *localhost* y el puerto 5000.
- ***api:app***: indica a gunicorn el módulo que debe ejecutar dentro de un archivo. Sigue la nomenclatura *archivo:módulo*. En este caso se utiliza el módulo app que está dentro del archivo *api.py*.

Al ejecutar el comando descrito anteriormente se instancia un servidor *gunicorn* con la API Rest desarrollada con Flask, mostrando en la consola todo el *log* de lo que va sucediendo durante la ejecución. También se puede utilizar en Linux el carácter *&* para indicar que ejecute el proceso en segundo plano y no se quede utilizando la consola. Una vez se haya llegado a este punto, la API estaría totalmente lista para atender peticiones.

En el siguiente apartado del presente anexo se explicará cómo desplegar el bot de Telegram desarrollado a modo de ejemplo como cliente para que pueda realizar las operaciones descritas en el apartado de desarrollo a través de la API Rest instanciada.

A.3. Despliegue del Bot

En este apartado se explicará cómo desplegar el bot de Telegram. Se trata de una tarea realmente sencilla, pues solamente se necesita instalar la librería *python-telegram-bot* mediante *pip*, tal y como se explicaba en el primer apartado de este anexo; la dirección y el puerto donde está escuchando la API Rest y un *token* que genera el BotFather de Telegram[41] para poder hacer uso de la API que ofrece este servicio de mensajería instantánea.

En primer lugar, destacar que no es necesario solicitar el *token*, pues en el código que se entrega junto al documento cuenta ya con un *token* otorgado por el Bot Father mencionado anteriormente, aunque un token solamente pertenece a una instancia, es decir, no puede haber dos instancias del bot con el mismo *token*. También, si se desea ejecutar en un lugar diferente a la API, es preciso cambiar la URL. Esta URL se encuentra en el código, en la línea 12, en una variable llamada *DIRECCION*. También se puede configurar la cámara utilizada para el comando */abrir_puerta*, pasando la ruta de una cámara IP o especificando un número correspondiéndose con las cámaras conectadas al

equipo donde esté corriendo este bot en la línea 30 del código fuente, empezando desde 0. La cámara que se utiliza por defecto es la primera conectada al ordenador.

Con todos los parámetros anteriores configurados como se desea, se ejecuta el siguiente comando, en el directorio donde se encuentra el programa, para instanciar el bot que será accesible – en este caso– por *@tfgcristianbot*:

```
$ python3 bot.py &
```

Una vez realizadas todas las acciones anteriores, tanto de la API Rest como las de este apartado, el proyecto estaría correctamente desplegado y listo para utilizarse.

Bibliografía

[1] Github. https://github.com/, visitado en Junio de 2018.

[2] https://blog.adext.com/es/machine-learning-guia-completa, visitado en Mayo de 2018.

[3] https://www.raona.com/machine-learning-tipos-machine-learning/, visitado en Mayo de 2018.

[4] Kevin Murnane. What is Deep Learning and How Is It Useful? https://www.forbes.com/sites/kevinmurnane/2016/04/01/what-is-deep-learning-and-how-is-it-useful/#63ae57f0d547.

[5] Jürgen Schmidhuber. Deep Learning in neural networks: An overview, 2015.

[6] Deep Learning Use Cases. https://deeplearning4j.org/use_cases, visitado en Mayo de 2018.

[7] Florian Schroff, Dmitry Kalenichenko, and James Philbin. FaceNet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015.

[8] Torch. http://torch.ch/, visitado en Mayo de 2018.

[9] DeepFace: Closing the Gap to Human-Level Performance in Face Verification, howpublished="https://research.fb.com/publications/deepface-closing-the-gap-to-human-level-performance-in-face-verification/" year = visitado en Mayo de 2018.

[10] https://www.baseapp.com/deepsight-image-recognition-sdk/, visitado en Mayo de 2018.

[11] FaceAPI: Software de reconocimiento facial. https://azure.microsoft.com/es-es/services/cognitive-services/face/, visitado en Mayo de 2018.

[12] MySQL Differences from Standard SQL. https://dev.mysql.com/doc/refman/8.0/en/differences-from-ansi.html, visitado en Junio de 2018.

[13] Ventajas y desventajas de MySQL. http://abcarticulos.info/article/ventajas-y-desventajas-de-mysql, visitado en Junio de 2018.

[14] https://docs.oracle.com/database/121/SQLRF/ap_standard_sql.htm#SQLRF019, visitado en Junio de 2018.

[15] Ventajas y desventajas de Oracle DBMS. http://oraclebddepn.blogspot.com/2013/05/ventajas-y-desventajas.html, visitado en Junio de 2018.

[16] Caffe. http://caffe.berkeleyvision.org/, visitado en Mayo de 2018.

[17] Qué es CUDA. http://www.nvidia.es/object/cuda-parallel-computing-es.html, visitado en Junio de 2018.

[18] What is HDF5? https://support.hdfgroup.org/HDF5/whatishdf5.html, visitado en Junio de 2018.

[19] Dlib. http://dlib.net/, visitado en Mayo de 2018.

[20] TensorFlow. https://www.tensorflow.org/, visitado en Junio de 2018.

[21] Theano. http://deeplearning.net/software/theano/, Mayo, 2018.

[22] Keras. https://keras.io/, visitado en Junio de 2018.

[23] OpenCV. https://opencv.org/, visitado en Junio de 2018.

[24] YOLO: Real-Time Object Detection. https://pjreddie.com/darknet/yolo/, visitado en Junio de 2018.

[25] SimpleCV. http://simplecv.org/, visitado en Junio de 2018.

[26] Flask. http://flask.pocoo.org/, visitado en Junio de 2018.

[27] Gunicorn. http://gunicorn.org/, visitado en Mayo de 2018.

[28] Express.js. http://expressjs.com/es/, visitado en Junio de 2018.

[29] Java Jersey. https://jersey.github.io/, visitado en Junio de 2018.

[30] Ranking Popular Deep Learning Libraries for Data Science. https://www.kdnuggets.com/2017/10/ranking-popular-deep-learning-libraries-data-science.html, 2017.

[31] https://ml4a.github.io/ml4a/es/neural_networks/, visitado en Junio de 2018.

[32] https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax?hl=es-419, visitado en Junio de 2018.

[33] http://deeplearning.net/software/theano/library/tensor/nnet/nnet.html#theano.tensor.nnet.nnet.categorical_crossentropy, visitado en Junio de 2018.

[34] Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent. https://goo.gl/4TKo7z, visitado en Junio de 2018.

[35] Geoffrey E. Hinton, Nitish Srivastava, and Kevin Swersky. Lecture 6a-overview of mini-batch gradient descent. *COURSERA: Neural Networks for Machine Learning*, 2012.

[36] https://keras.io/callbacks/, visitado en Junio de 2018.

[37] python-telegram-bot. https://github.com/python-telegram-bot/python-telegram-bot, visitado en Junio de 2018.

[38] Python Decorators. https://wiki.python.org/moin/PythonDecorators, visitado en Junio de 2018.

[39] Mala Sundaram and Ambika Mani. Face Recognition : Demystification of Multifarious Aspect in Evaluation Metrics. In *Face Recognition - Semisupervised Classification, Subspace Projection and Evaluation Methods*. 2016.

[40] Postman. https://www.getpostman.com/, visitado en Junio de 2018.

[41] Botfather of Telegram. https://core.telegram.org/bots#6-botfather, visitado en Junio de 2018.