UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Diseño de un sistema de identificación de personas

Presentado por Víctor de Castro Hurtado
en Universidad de Burgos — 31 de mayo
de 2018
Tutor: César Represa Pérez

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. César Represa Pérez, profesor del departamento de Ingeniería Electromecánica, área de Tecnología Electrónica.

Expone:

Que el alumno D. Víctor de Castro Hurtado, con DNI 71289378G, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado Diseño de un sistema de identificación de personas.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 31 de mayo de 2018

Vº. Bº. del Tutor:

D. César Represa Pérez

Índice general

Índice de figuras

Índice de tablas

Introducción

1.1. El problema

Las tecnologías avanzan continuamente, al igual que hacen las amenazas que acompañan a dichas tecnologías. Por ello, y para mantener una sociedad estable, la seguridad debe ir ligada a estos avances.

Últimamente escuchamos hablar mucho sobre ataques terroristas, tiroteos, asesinatos, etc., y ciertas medidas que se están tomando en algunos países (como EEUU) para intentar impedir dichos ataques antes de que sucedan.

Pero estas medidas llevan un coste asociado: la pérdida de privacidad. Es un coste que mucha gente no está dispuesta a pagar, y es un tema muy interesante que podría dar para una reflexión más larga, pero no es el objetivo del proyecto, de modo que asumiremos que no hay discrepancias en cuanto a aumentar la seguridad en lugares públicos.

Otro hecho es que, a día de hoy, todo está conectado, desde nuestros teléfonos móviles, hasta toda nuestra información en las redes sociales, así como los nuevos vehículos de conducción automática, etc. Por ello, y como indicábamos antes, se pierde privacidad, pero también se gana en seguridad: al estar todo conectado, es más fácil controlarlo.

1.2. La solución

Con este proyecto se pretende aumentar la seguridad en lugares públicos (como pueden ser aeropuertos o centros comerciales) frente a ataques terroristas o cualquier amenaza similar. Para ello, se parte de una base de datos que contiene imágenes de gente en una lista de **busca y captura**. A partir de estas imágenes se pretende identificar a las personas de la base de datos que se encuentren en dichos lugares públicos.

Como se mencionaba antes, al estar todo conectado, encontrar a esta gente entre la multitud o incluso entre las redes sociales es más sencillo que nunca, y en este proyecto se pretende diseñar un sistema de **identificación de personas** a través de un programa de reconocimiento facial, desarrollado en *Python* y utilizando técnicas de **Machine Learning**.

Y, ¿por qué machine learning? 3.1

Bueno, siempre ha estado ahí, aunque ha sido cuando lo han empezado a usar las grandes compañías (*Google*, *Apple*) cuando realmente se ha visto su verdadero potencial. Consiste, en resumidas cuentas, en que ya no tenemos que decirle al ordenador lo que tiene que hacer en cada ocasión con instrucciones, sino que se le indica un modelo de comportamiento que debería seguir (como se enseña a los niños pequeños), y es él el encargado de actuar en cada caso como entienda que es mejor. Es una técnica muy útil y versátil, y puede ser utilizado desde reconocimiento de objetos en imágenes (como es nuestro caso), a aprender a formar frases completas y con sentido, como es el caso de la nueva funcionalidad de *Google*: **Google Duplex** [43].

Una de las muchas aplicaciones que tiene el *machine learning* es el *facial recognition* que vamos a usar en nuestro caso, el cual tiene a su vez muchas aplicaciones, cada día más utilizadas, destacando las relacionadas con la **seguridad**, como la utilizada en los aeropuertos para llevar un control de los pasajeros, o en las cámaras públicas de las ciudades, para obtener información sobre atracos, ataques terroristas, etc.

También destacan (aunque no nos interesan en la realización de este proyecto) los usos en reconocimiento y seguimiento de clientes por parte de empresas y tiendas, así como el reciente *desbloqueo facial* de los teléfonos inteligentes como los de *Apple*.

En el siguiente enlace [7] se pueden ver distintas aplicaciones desarrolladas por *tensorflow* (herramienta para desarrollar aplicaciones de machine

learning) [40], no sólo para el reconocimiento facial, sino para el reconocimiento de imágenes en general.

1.3. Funcionamiento general

Como hemos mencionado, el proyecto pretende servir como forma de identificación de personas que, teniendo una imagen de la persona en cuestión previamente obtenida en la base de datos, y su información básica (nombre, edad, lugar de nacimiento y profesión), se puede identificar a dicha persona cuando se pone delante de la cámara utilizando técnicas de machine-learning y computer vision.

Aplicación real a gran escala

Para ello, se deberían tener las imágenes de dichas personas en busca y captura almacenadas en la base de datos, todas ellas con las mismas características (tomadas de un registro oficial, como pueden ser imágenes de DNI, registros de su paso por comisaría en otras detenciones, organizaciones internacionales, etc).

Teniendo las personas que queremos encontrar ya introducidas en nuestra base de datos, se procedería a ejecutar un **rastreo**, ejecutando el programa en un **servidor** con capacidad suficiente como para analizar cada rostro de todas las imágenes en tiempo real, sobre las cámaras de ayuntamientos, bancos, semáforos, comisarías, aeropuertos, estaciones de autobús o tren o complejos deportivos de la ciudad.

Por supuesto, este sistema de reconocimiento a **gran escala** y en **tiempo real** (recordemos que está recopilando información de todas las cámaras disponibles en la ciudad, a la vez que está obteniendo las características de los rostros que encuentra en cada una de ellas, comparando dichas características con las obtenidas al entrenar previamente la red), tiene un **alto coste computacional** y de recursos, de manera que sería necesario ejecutar el programa principal en un servidor con unos requisitos técnicos muy altos, además de una infraestructura adecuada para poder transmitir toda esa información sin retrasos (lo que implica, de nuevo, unos altos requisitos técnicos).

Si alguna de estas personas aparece en alguna de las cámaras, el sistema le reconoce y muestra una interfaz con la imagen capturada desde la cámara y la imagen de la base de datos con la que la relaciona, mostrando además la cámara con la que ha realizado el reconocimiento, para que sea más sencillo localizar a esta persona en la zona de la ciudad que corresponda.

Estamos hablando de una aplicación que debería ser llevada a cabo **a nivel local** (en un municipio o ciudad no demasiado grande), porque, como hemos comentado, cuanto mayor sea la zona a cubrir (y por lo tanto el número de cámaras utilizadas), mayores serán los requisitos técnicos necesarios.

Otra posible aplicación que se podría obtener de este proyecto sería el buscar personas desaparecidas. Utilizando el mismo método, lo único que habría que variar sería la base de datos utilizada: de criminales a gente desaparecida.

Lo mismo que se podría adaptar a otros ámbitos simplemente teniendo una base de datos diferente para cada uno de los casos. En ese sentido, el **coste de adaptación** a la variedad de casos de usos sería **mínimo**, siendo el coste inicial el mayor de todos: un servidor principal que se encargue de ejecutar el programa, recopilando los datos obtenidos de todas las cámaras.

Aplicación real con los recursos disponibles

En la realización de este proyecto no se ha llegado al nivel de desarrollo descrito en el apartado anterior, primero porque no se disponía de los medios necesarios para cumplir con los requisitos del hardware (el servidor principal con requisitos técnicos elevados que comentábamos anteriormente), y segundo porque no se disponía de una base de datos suficientemente amplia como para mostrar esta funcionalidad completa que se mencionaba en párrafos anteriores.

Por lo tanto, se ha optado por describir sus posibles y futuras implicaciones en esta memoria, mientras se implementaba una primera versión del proyecto, acorde al presupuesto, tiempo y recursos de los que se disponía.

En esta primera versión se tiene una base de datos con unas cuantas imágenes de personajes famosos (representaría en la aplicación final a los criminales), entre los que se incluye una foto nuestra para poder obtener un resultado positivo en el análisis en tiempo real.

Con esta base de datos improvisada, se entrena la red, identificando las características de las personas que tenemos, almacenando dichas características para evitar entrenar la red cada vez que se trate de identificar un rostro (ahorrando tiempo y recursos).

Tras esto, ejecutamos la parte del programa encargada del reconocimiento propiamente dicho, por lo que nos colocamos delante de la cámara y tomamos una captura (esta parte también se ha visto simplificada con respecto a la aplicación final en la que se tomarían dichas capturas con varias cámaras, distribuidas a lo largo de la ciudad).

Por último, el programa reconoce automáticamente las características del rostro de la imagen capturada, compara dichas características con las que teníamos guardadas de los famosos, y nos da una estimación de la persona a la que más nos parezcamos.

En una aproximación más real y comercial, bastaría con adaptar los parámetros de comparación y muestreo de los resultados para evitar mostrar los resultados que no superen un umbral, de tal manera que, si no coincide en un alto porcentaje con la persona que estamos buscando, no muestre nada, evitando de esta manera falsos positivos.

1.4. Estructura del repositorio

La estructura que sigue nuestro proyecto y que se puede encontrar en Github 1.6 es la siguiente:

- Fichero **run.bat**

  Este es el fichero que deberemos ejecutar para ver como funciona el programa sin necesidad de importar el proyecto entero en un editor de python. Al darle doble click (depende de los permisos que tengamos, puede ser necesario ejecutarlo como administrador), se encargará de ejecutar el programa desde el principio.

- Carpeta **Other**

  En esta carpeta se encuentra el material ajeno al proyecto en python, y que, por lo tanto, no afecta a la ejecución del programa. En dicha carpeta se encuentra:

  - Carpeta **Images**

    Contiene diversas imágenes relacionadas con el desarrollo del proyecto, explicaciones sobre algunos aspectos cuya descripción con palabras podía no quedar demasiado clara, etc.

    La mayoría de estas imágenes se han realizado de forma 'casera' por nuestra cuenta, aunque pueden encontrarse también algunas imágenes obtenidas de internet (en cuyo caso se especificará su orígen).

    - Carpeta **ClassDiagram**

      Esta carpeta contiene los diagramas de clases, tanto los que representaban la estructura inicial del proyecto como el resultado final. En ellos se incluye también la estructura de paquetes, cómo se relacionan entre ellas, etc. La explicación

sobre ello se puede encontrar en el Anexo C-Diseno del documento anexos.pdf.

- Carpeta **flowCharts**

  Esta carpeta contiene los diagramas de flujo del programa, representando cuales son los pasos que sigue, posibles ramas y desenlaces que puede tener. Como en el caso de los diagramas de clases, se incluyen los diagramas con la idea original y el resultado final. La explicación sobre ello se puede encontrar en el Anexo C-Diseno del documento anexos.pdf.

- Carpeta **theoric concepts**

  Esta carpeta contiene imágenes relacionadas con los conceptos teóricos de la sección 3.- Conceptos teóricos 2.3.

- Carpeta **results**

  Esta carpeta contiene imágenes sobre la ejecución del programa, para poder observar cómo sería el resultado tanto si se ejecuta sin problemas, pero no identificando a la persona que tiene delante de la cámara, como si se ejecuta sin problemas identificándola, incluso posibles fallos.

- Fichero **TFG-FacialRecognition-Memoria.pdf**

  Contiene esta memoria en formato .pdf.

- Fichero **TFG-FacialRecognition-Anexos.pdf**

  Contiene los anexos de la memoria, en formato .pdf.

- Carpeta comprimida **TFG-FacialRecognition.zip**

  Contiene esta memoria y los anexos, divididos en diferentes ficheros .tex.

- Fichero **videoExplicativo.mp4**

  Un vídeo que se ha realizado explicando el funcionamiento general del programa.

- Carpeta **sources**

  En esta carpeta se puede encontrar material que afecta a la ejecución del programa, pero que no es código ejecutable como tal. En dicha carpeta encontramos:

  - Carpeta **dataset**

    En esta carpeta podemos encontrar las imágenes originales que guardamos al principio en nuestra base de datos. Como se puede observar, son imágenes de cuerpo entero la mayoría de ellas, por lo tanto no están ajustadas a la posición de la cara de la persona, por lo que necesitamos crear (interna y automáticamente) la siguiente carpeta:

  - Carpeta **facesDataset**

    Contiene las imágenes anteriores, pero exclusivamente con la cara que haya podido encontrar en la imagen, recortando dicha imagen para eliminar elementos ajenos al rostro que puedan dificultar el reconocimiento.

  - Carpeta **xml**

    Contiene dos ficheros .xml que son los encargados de encontrar las partes identificables de un rostro en la imagen. Dichos ficheros se proporcionan gratuitamente con fines educativos en un repositorio [29].

  - Carpeta **recognizer**

    Contiene ficheros que utilizamos durante la ejecución del programa para visualizar datos sobre la persona reocnocida o cargar la

imagen correcta de la base de datos a partir de un diccionario.

  - Fichero **dictionary-ID-labels.txt**

    Este fichero se crea automáticamente tras entrenar la red, y contiene el diccionario que relaciona una ID automática a cada imagen de la que ha conseguido obtener un rostro. Se podría evitar usar este fichero si se utilizara directamente el diccionario en el programa, pero eso supondría tener que entrenar la red todas las ejecuciones, y si no se han añadido imágenes nuevas es tiempo y recursos perdidos, de modo que se ha optado por crear un fichero para que el diccionario persista a cada ejecución.

  - Fichero **info.txt**

    Este fichero se ha creado sólo para mostrar información de la persona que se haya reconocido durante la ejecución. Para mostrar esta información consultar el anexo E-Manual-usuario. Esta información se puede actualizar a mano actualmente, aunque en un futuro se podría incluir en la base de datos para evitar accesos no permitidos.

  - Fichero **trainedData.yml**

    Este fichero contiene las principales características de los rostros detectados cuando entrenamos la red. Es el resultado de dicho entrenamiento, y por lo tanto se crea automáticamente. Las características que obtiene son las mismas que las que se pueden encontrar en los ficheros .xml de dicha carpeta.

- Fichero **default.png**

  Esta imagen es la que carga el programa por defecto cuando no se ha conseguido superar el umbral de reconocimiento de un rostro en una imagen. Normalmente se conseguirá superar dicho umbral, aunque el reconocimiento falle, pero en caso de que no se hayan podido encontrar rostros, no se haya entrenado la red, o se le pase un fichero que no sea una imagen, el programa cargará esta

imagen por defecto.

- Carpeta **src**

  En esta carpeta se encuentra el código de nuestro programa. Está dividido en diferentes clases que vemos a continuación:

  - Fichero **Main.py**

    Contiene el método principal de la ejecución del programa. Es donde se le pregunta al usuario si quiere entrenar la red, seleccionar una imagen desde fichero o desde la cámara, llama a todos los demás métodos de otras clases para comparar las imágenes, generar la interfaz y mostrar resultados. Se pueden ver la estructura en el anexo D-Manual-programador, y el flujo de programa en el anexo C-Diseno.

  - Fichero **Util.py**

    Contiene todos los métodos de utilidad, desde pedir una simple cadena de texto por teclado hasta inicializar la cámara y guardar la imagen que queramos para compararla, pasando por cargar los diferentes ficheros .xml y .txt que usamos o mostrar los menús por pantalla. Se pueden ver todos los métodos en el anexo D-Manual-programador.

  - Fichero **GUI.py**

    Contiene todo lo relacionado con la interfaz gráfica, desde el panel central, donde se muestra toda la información y las imágenes obtenidas, hasta el menú que nos permite seleccionar una imagen desde fichero en vez de desde la cámara. Dicha interfaz de ha realizado con tkinter. Se puede ver cada uno de los métodos y su funcionalidad en el anexo D-Manual-programador.

  - Fichero **CompareImages.py**

Realiza la comparación entre la imagen que acabamos de obtener (ya sea mediante captura con la cámara o desde fichero), con todas las de la base de datos. Se puede ver el funcionamiento exacto de esta función en el anexo D-Manual-programador.

- Fichero **trainer.py**

  Esta clase se encarga exclusivamente de entrenar la red a partir de unas imágenes fuente, que se encuentran en la carpeta faces-Dataset. La funcionalidad completa se puede observar en el anexo D-Manual-programador.

- Fichero **Camera.py**

  Esta clase tiene métodos básicos a la hora del desarrollo del programa principal, como capturar imágenes desde la cámara, localizar los rostros o inicializar la propia cámara.

- Fichero **README.txt**

  Fichero con información general sobre el proyecto, instrucciones básicas y modo de empleo.

1.5. Estructura de la memoria

- **Introducción**: descripción del problema a resolver y la solución propuesta. Estructura de la memoria y del proyecto, así como los materiales adjuntos.
- **Objetivos del proyecto**: exposición de los objetivos que persigue el proyecto, tanto técnicos como personales.
- **Conceptos teóricos**: explicación de los conceptos teóricos clave para la comprensión del proyecto.
- **Técnicas y herramientas**: técnicas metodológicas y herramientas utilizadas para el desarrollo del proyecto.
- **Aspectos relevantes del desarrollo**: explicación del desarrollo del proyecto, destacando los aspectos más relevantes.
- **Trabajos relacionados**: proyectos relacionados.
- **Conclusiones y líneas de trabajo futuras**: conclusiones obtenidas y posibilidades de mejora del proyecto.

Junto a la memoria se proporcionan los siguientes anexos:

- **Plan del proyecto software**: planificación temporal y estudio de viabilidad del proyecto.
- **Especificación de requisitos del software**: fase de análisis del proyecto, objetivos generales, requisitos funcionales y no funcionales.
- **Especificación de diseño**: fase de diseño, tanto del software como de los datos.
- **Manual del programador**: aspectos relevantes relacionados con el código fuente, localizado en la carpeta *src*.
- **Manual de usuario**: guía de usuario con instrucción y aspectos destacables que puedan facilitar el correcto manejo de la aplicación.

1.6. Material complementario

- Repositorio de *Github* https://github.com/victorcas04/TFG-FacialRecognition

- CD/DVD El contenido del CD es el directorio completo del proyecto, con todos los subdirectorios y ficheros comentados en la sección anterior 1.4, entre los que se incluye el código fuente de la aplicación, la memoria con sus anexos, y un vídeo explicativo del funcionamiento básico del programa.

Objetivos del proyecto

2.1. Objetivos generales

- Cargar una imagen leída desde una cámara.
- Comparar dicha imagen con otras imágenes almacenadas.
- Mostrar la imagen almacenada con mayor coincidencia.

2.2. Objetivos técnicos

- Obtener, manipular y guardar imágenes con las herramientas *OpenCV* 4.3 y *Pillow* 4.3.

- Reconocer un rostro dentro de la imagen utilizando un fichero 3.3 con las principales características de una cara.

- Entrenar una red mediante técnicas de *machine learning* y *computer vision* utilizando como entrada las imágenes de la base de datos, y como herramienta *OpenCV* 4.3 y una técnica *LBPH* 3.3.

- Hacer que se reconozca el rostro de la persona de la imagen capturada a partir de los resultados de la red entrenada, comparando características de rostros.

- Crear una interfaz adecuada para mostrar los resultados finales utilizando como herramientas principales *Pillow* 4.3 y *tKinter* 4.3.

- Realizar el informe con la herramienta *LaTex* 4.3.

2.3. Objetivos personales

- Aplicar varios de los conocimientos obtenidos durante la carrera.

- Familiarizarme más con el lenguaje de programación *Python*, y sus herramientas más comunes.

- Ver y conocer de primera mano aquello de lo que habla todo el mundo en el mundo de la última tecnología (*machine-learning*).

- Desarrollar un proyecto que pueda servir de carta de presentación a la hora de buscar empleo o empresas que estén interesadas en seguir adelante con dicho proyecto.

- Aprender a desarrollar una interfaz para una aplicación desarrollada en *Python*, aunque sencilla, que permita ir profundizando poco a poco en la 'puesta en escena' de nuestro programa.

Conceptos teóricos

Algunos de los conceptos relacionados con el presente proyecto son:

3.1. IA

Inteligencia Artificial

La inteligencia artificial (IA) según *searchdatacenter* [32], es 'la simulación de procesos de inteligencia humana por parte de máquinas, especialmente sistemas informáticos. Estos procesos incluyen el aprendizaje (la adquisición de información y reglas para el uso de la información), el razonamiento (usando las reglas para llegar a conclusiones aproximadas o definitivas) y la autocorrección.'

Aprendizaje Automático

El aprendizaje automático (o más conocido por su nombre en inglés: Machine Learning). es la parte de la inteligencia artificial que consiste en conseguir que un ordenador actúe de manera automática, sin necesidad de programarlo para ello.

Analítica predictiva

La analítica predictiva es la ciencia que analiza los datos para intentar predecir datos futuros relacionados. En nuestro caso, a partir de unos datos originales, se analizan mediante el aprendizaje automático para crear nuevos

modelos predictivos.

Aprendizaje profundo

El aprendizaje profundo (o más conocido por su nombre en inglés: Deep Learning), es una rama del aprendizaje automático que automatiza la analítica predictiva.

En el siguiente artículo de *Christian S. Perone* [31], se puede encontrar un ejemplo de una red convolucional desarrollada en python.

Visión Artificial

La visión artificial [44] (o más conocido por su nombre en inglés: Computer Vision) pretende simular el comportamiento del ojo humano, permitiendo adquirir, procesar, analizar y comprender las imágenes mediante la clasificación y segmentación de las mismas utilizando técnicas de machine learning.

Las grandes empresas de tecnología están desarrollando su propio software para aprovechar al máximo sus aplicaciones, una de ellas, y de las que mejores resultados da, es *Google* [17].

3.2. Redes neuronales

Red neuronal artificial

Las redes neuronales artificiales [48] son un modelo computacional basado en nodos y enlaces, donde **cada nodo corresponde con una neurona**, y los enlaces corresponden a las relaciones entre dichos nodos. Cada nodo almacena información independiente del resto de nodos, de manera que nuestro sistema se divide en función de la información independiente que tenga.

Dados unos **datos de entrada** (nodos *input*), se forman relaciones entre ellos para establecer los aspectos comunes y las diferencias.

De estas relaciones nos quedamos con los nodos intermedios de la red (nodos *hidden*), que son los que no ve el usuario (programador), y que puede haber tantas capas de nodos *hidden* como hagan falta.

Por último, la última de estas capas se conforma de los nodos de salida que vamos a obtener como **resultado del entrenamiento** de la red (nodos *output*).

Este proceso se puede observar en la imagen 3.1.

El objetivo de la red neuronal es resolver los problemas de la misma manera que el cerebro humano, aunque de forma más abstracta y con variantes de entre miles a millones de neuronas (nodos).

Algo importante cuando hablamos de redes neuronales, inteligencia artificial y machine learning, es que **no se puede garantizar nunca su grado de éxito** después de realizar el auto-aprendizaje. Para obtener unos resultados más o menos homogéneos, se necesitan miles y miles de ejecuciones.

Red neuronal convulacional

Las redes neuronales convolucionales [49] son un subtipo de las redes neuronales artificiales, que se basan en las versiones biológicas de los perceptrones multicapa (Multi Layer Perceptron en inglés) [47], y consisten en **varias capas** o *layers*, en cada una de las cuales existen una serie de nodos con información. Estos nodos se conectan con cada uno de los nodos

de la capa siguiente (a diferencia de las redes neuronales normales, que simplemente crean un nodo de la capa siguiente a partir de dos de la capa anterior), quedando **todos los nodos interconectados entre una capa y otra**. Este tipo de redes son especialmente útiles en sistemas de computer vision y deep learning, ya que los nodos se retroalimentan entre si, dando lugar a mejores soluciones.

Este proceso se puede observar en la siguiente imagen 3.2. Para obtener más información sobre este tipo de redes se puden consultar las siguientes referencias [34], [21].

Tipos de Entrenamiento

Los diferentes tipos de entrenamiento de una red son:

- Aprendizaje supervisado

  En este tipo de aprendizaje los datos tienen etiquetas, las cuales se usan para obtener nuevos patrones relacionados con los datos de dichas etiquetas.

- Aprendizaje no supervisado

  En este tipo de aprendizaje los datos no tienen etiquetas, de manera que los nuevos patrones se obtienen y clasifican en función de similitudes y/o diferencias con el resto de datos.

- Aprendizaje por refuerzo

  En este sistema de aprendizaje, como en el no supervisado, no hay etiquetas, pero a diferencia de este, tras realizar un modelo o patrón, el sistema recibe retroalimentación para ajustar dicho patrón.

3.3. Reconocimiento Facial

El reconocimiento facial [9], [13] es una de las principales aplicaciones de la visión artificial, y consiste en localizar y reconocer rostros de gente en imágenes.

El proceso de este reconocimiento se compone de dos partes (para la información completa consultar el siguiente artículo [14]):

Detección de rostros

En esta primera parte del proceso se identifican los pixeles de la imagen que pertenezcan a un rostro. Para realizar dicha identificación, hay varios algoritmo, aunque uno de los más utilizados y extendidos es el llamado *Haar Cascade face detection* [2]. Todas las versiones de este algoritmo se pueden encontrar en el siguiente repositorio [29].

Hay muchas aplicaciones [23], [22] que se aprovechan de esta detección de rostros sin llegar a usar el reconocimiento facial, ya que no supone tanto trabajo implementar esta primera parte como seguir con el proceso.

Reconocimiento de rostros

En esta segunda parte se utilizan diversas técnicas para evitar los problemas derivados de la calidad de la imagen (iluminación, posición de la persona, cambios en los rasgos faciales, etc) como pueden ser las funciones que realizan cortes o *thresholds* de nuestra imagen a partir de ciertos valores umbrales, y se obtiene como resultado una segunda imagen más *estandarizada*, la cual comparamos (usando en nuestro caso los CascadeClassifier [1]) con las otras imágenes que teníamos preparadas para ser comparadas, y como resultado final obtenemos una de estas imágenes, que será la que contenga el rostro que queríamos reconocer en la primera imagen (o la que más se acerque).

Eigenvectores

Un eigenvector o valor propio de una imagen es, según wikipedia [51], ün vector no nulo que, cuando es transformado por el operador, da lugar a un múltiplo escalar de sí mismo, con lo que no cambia su dirección."

Esta definición a nosotros no nos interesa mucho, aunque si nos interesa su aplicación, que viene a significar lo siguiente:

De una imagen, obtenemos todos sus píxeles, y hacemos que cada uno de ellos represente un elemento de una matriz de tamaño MxN (tamaño de la imagen). Empezamos por un píxel (generalmente al azar), y realizamos la operación que hemos visto en la definición más arriba con cada uno de los píxeles en los alrededores.

De esta manera estamos **comparando su dirección con la de cada uno de los píxeles cercanos**, para saber cuales de ellos hacen que cambie su dirección.

Si todos los píxeles en los alrededores mantienen la dirección del pixel actual, significa que todos ellos pertenecen al mismo elemento dentro de la imagen.

Si hay alguno que hace cambiar de dirección el eigenvector del pixel actual, significa que esos dos píxeles pertenecen a elementos diferentes de la imagen.

De esta manera se pueden obtener las diferentes características de una imagen a partir de los eigenvectores.

Hay multitud de aplicaciones de los eigenvectores relacionados con imágenes por internet, en la siguiente referencia se muestran algunas de ellas [18].

Algoritmos de Reconocimiento Facial

Los diferentes algoritmos que se barajaban para realizar el reconocimiento facial fueron:

- Eigenfaces

Las eigencaras o eigenfaces [53] es el nombre que se les da a los eigenvectores cuando son usados en el reconocimiento facial de las personas con computer vision.

Es la forma más básica y una de las primeras técnicas conocidas para el reconocimiento facial, que se empezó a usar a partir de 1991.

- Fisherfaces

  Las fisherfaces [27] se basan en el mismo modelo que las eigenfaces, aunque en este caso se tiene también en cuenta la luz y los espacios entre características del rostro, dando más importancia a las expresiones faciales que a los propios rostros.

  A partir de 1997 se le empezó a dar un poco más de importancia a este método, aunque nunca acabó de tener mucho éxito.

- Local binary patterns

  Según *Kelvin Salton* [35], el algoritmo LBP (Local Binary Pattern) es un sistema simple pero eficaz, que consiste en etiqutar los píxeles de una imagen a partir del umbral que se le asigna a cada uno de sus vecinos. Esto significa que le da un valor a cada uno de los píxeles en función de la intensidad de color de los píxeles cercanos [33]. Para ello es necesario tratar la imagen en **escala de grises**.

  Si la intensidad de color de uno de los píxeles es mayor o igual, a ese pixel se le asigna un *1*, en caso contrario un *0*. A continuación, se juntan los 8 bits obtenidos de los 8 píxeles vecinos al pixel que estemos comparando y se forma un número en binario (el órden en que se leen los bits es irrelevante, mientras se use el mismo órden en toda la imagen, aunque lo más común suele ser empezar por una esquina y seguir el sentido de las agujas del reloj, aunque no hay ningún standard), que corresponderá con la nueva **intensidad de color** del pixel (de 0 -negro- a 255 -blanco-).

  Con esta nueva intensidad de color definida para todos los píxeles de la imagen, se puede obtener una relación entre ellos, saber qué pixeles están más relacionados entre ellos que con otros, etc. A partir de cada una de estas agrupaciones **se obtienen las características** que vamos a utilizar en nuestro sistema de reconocimiento facial.

Se puede observar todo el proceso en la imagen 3.3.

Se empieza a estudiar antes que las fisherfaces, aunque hasta 1996 no se pone de moda. A partir de entonces se ha convertido en el más utilizado por su sencillez y relativa eficacia.

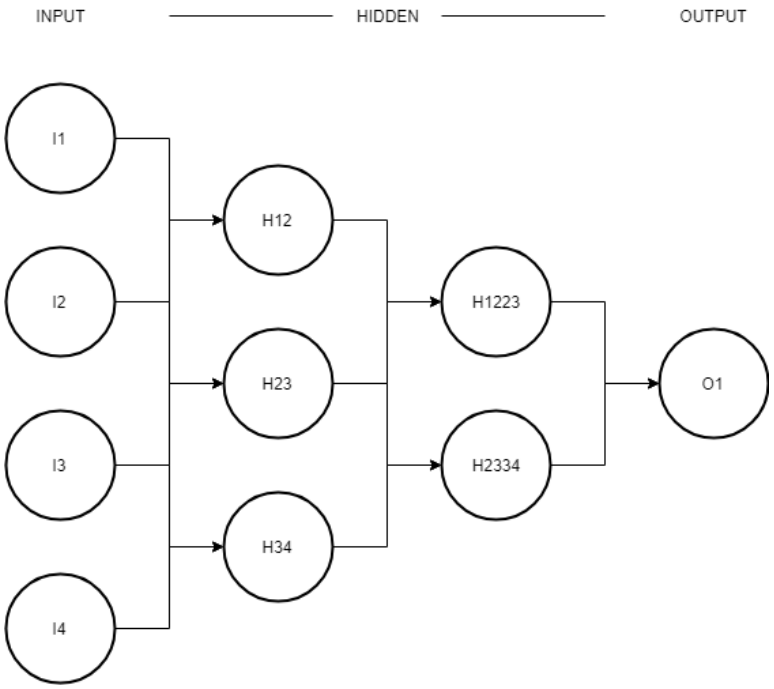

Figura 3.1: Red neuronal con las diferentes capas implicadas.

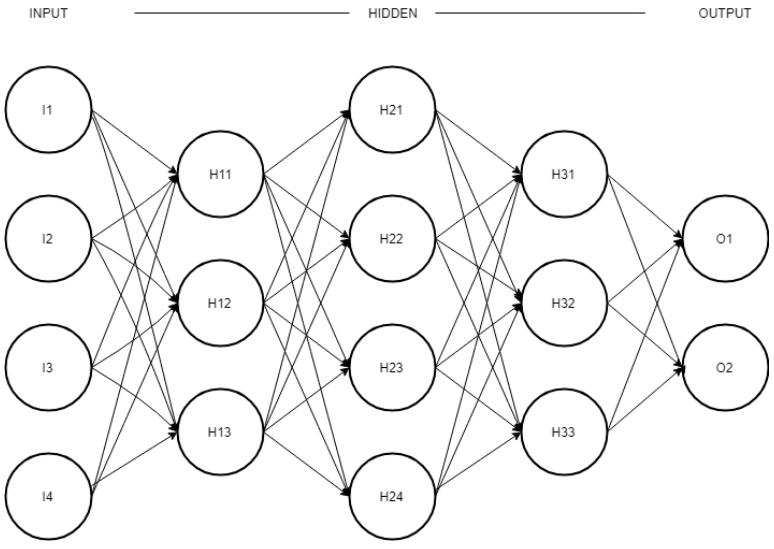

Figura 3.2: Red neuronal convolucional con las diferentes capas implicadas y cómo se relacionan los nodos de cada una entre si.

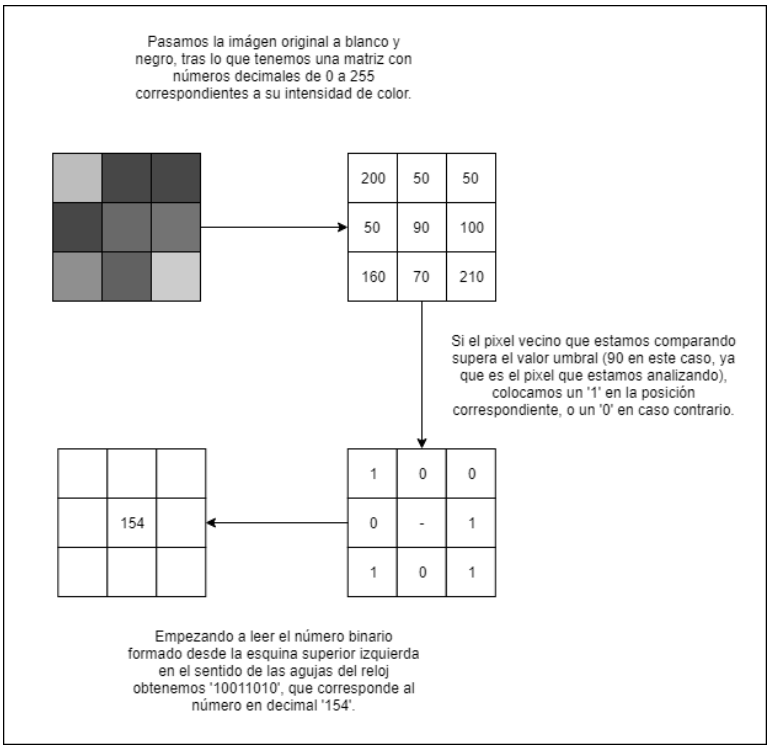

Figura 3.3: Proceso de agrupación e identificación de características en una imagen según el algoritmo LBPH.

Técnicas y herramientas

4.1. Técnicas metodológicas

Aunque nos hemos basado en un método de **desarrollo en cascada** [45], hemos incluido algunos elementos de la **metodología scrum** [50] como las reuniones mensuales.

En cuanto al desarrollo en cascada, hemos tomado la base de este método, ya que teníamos unos **requisitos iniciales**, que eran los objetivos primarios 2.1 que teníamos que cumplir, a partir de los cuales se hizo un primer **diseño del proyecto**.

Aquí es donde entra la metodología scrum, y es que incluimos la primera reunión con el tutor, para ver que estructura tenía el proyecto y que posibles cambios iniciales se podrían realizar.

Esta primera fase del proyecto (identificación de requisitos, planteamiento del proyecto y diseño de la estructura básica del mismo) duró aproximadamente un mes, desde principio de Enero hasta Mediados de Febrero.

A continuación, seguimos con el siguiente paso de la metodología en cascada: el **desarrollo e implementación**.

Si bien es cierto que el desarrollo ha continuado hasta la última semana de Mayo, casi todo el código se implementó desde mediados de Febrero a principios de Mayo, algo más de dos meses. A partir de esa fecha el desarrollo del código fue más de corregir algunos errores, mejorar la apariencia, etc.

Durante toda esta fase de desarrollo se tuvieron varias reuniones mensuales con el tutor para ver cómo iba avanzando el proyecto, lo que necesitaba ser

mejorado o cambiado, o asignar nuevas tareas hasta la próxima reunión (en nuestro caso la mayoría fueron semanales o quincenales, en vez de mensuales).

En el apartado *5_Aspectos_relevantes_del_desarrollo_del_proyecto* 4.3 se explican las fases del desarrollo de manera detallada, asignando las tareas o *issues* principales a cada uno de ellos.

Además, en los anexos correspondientes a estas fases (*A_Plan_proyecto* para la fase de Planteamiento del proyecto, *B_Requisitos* para la fase de Identificación de requisitos, *C_Diseño* para la fase de Diseño del proyecto y *D_Manual_programador* para la fase de Desarrollo) se explica y detalla cada una de ellas.

4.2. Patrones

A la hora de desarrollar el proyecto, no se hizo pensando en utilizar unos u otros patrones de manera intencionada, sino que se fueron añadiendo en función de las necesidades.

Al principio se utilizaron más patrones [24], como los **Adaptadores ??**, que utilizaban clases intermedias para comunicar dos elementos diferentes del proyecto (la base de datos con la interfaz, por ejemplo).

O en el caso de obtener las imágenes que íbamos a analizar, en una primera versión se utilizaba el patrón **Abstract Factory** 4.5, ya que existía una interfaz *CaptureImage* que instanciaba una clase concreta *CaptureImageFromFile* o *CaptureImageFromCamera* en función del orígen de la imagen.

Aunque posteriormente se unificaron varias clases, se hizo refactorización de muchos de los métodos anteriores, y se consiguió eliminar complejidad de código, referencias y clases, por lo que algunos de estos patrones ya no eran necesarios.

Al final, el patrón más utilizado y que más claramente podemos encontrar es el de **Singleton** 4.6, especialmente útil cuando queremos instanciar una nueva interfaz gráfica, que nos ayuda a asegurar que no se creen y sobreescriban varias interfaces (ya que la utilizamos de varias maneras diferentes, no nos interesa crear una interfaz para cada uno de estos usos, sino una común que vaya cambiando en función del uso que le queramos dar).

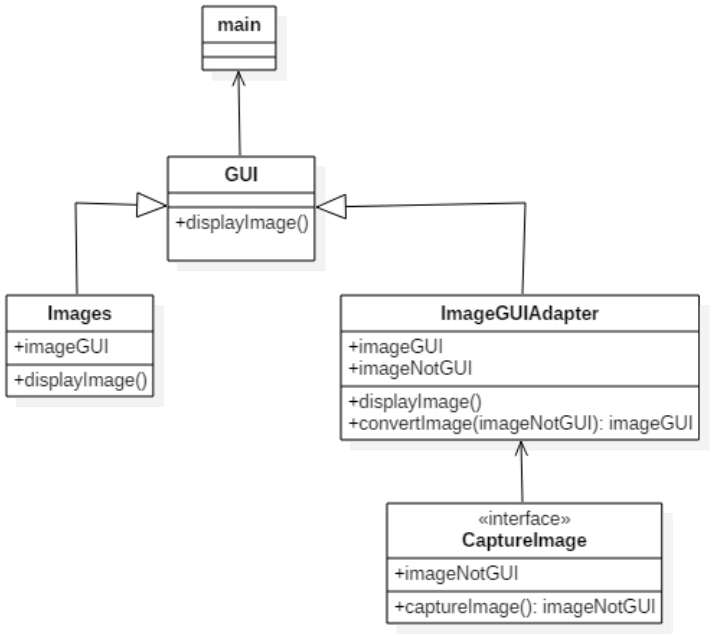

Figura 4.4: Ejemplo de patrón Adaptador utilizado al comienzo del desarrollo del proyecto.

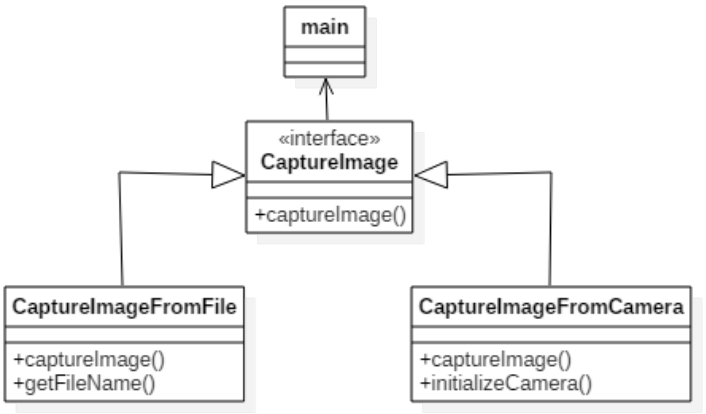

Figura 4.5: Ejemplo de patrón Abstract Factory utilizado al comienzo del desarrollo del proyecto.

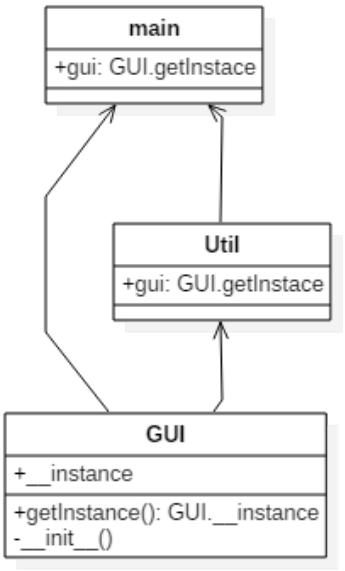

Figura 4.6: Ejemplo de patrón Singleton utilizado en la clase GUI.

4.3. Herramientas

Algunas de las herramientas que hemos utilizado para el desarrollo del proyecto han sido los siguientes:

Herramientas de desarrollo

- El lenguaje de programación ha sido *Python* [10].
- Para realizar la instalación del entorno virtual utilizado durante el desarrollo del trabajo se ha utilizado *Anaconda* [3].
- Herramienta *OpenCV* [30]
- *Pillow* para python [5]
- *tKinter* para python [11]
- Se han obtenido respuestas y soluciones a varios de los problemas encontrados durante la realización del proyecto en *stackoverflow* [19].
- El editor de texto utilizado ha sido *pycharm* [20].
- Se ha utilizado la aplicación *image.net* [39] para la obtención de algunas imágenes de muestra.

Herramientas de planificación y diseño

- Para las tareas semanales/mensuales y llevar un control de las versiones del proyecto se ha utilizado *Github* [15] y su cliente nativo para el escritorio de *Windows* Github Desktop [16].
- Para los diagramas de clases se ha usado la herramienta gratuita *starUML* [28].

- Para los flujogramas y otras imágenes de esquemas y diagramas se ha utilizado la herramienta *draw.io* [26].

- Para la memoria se han utilizado la herramienta *LaTex* [46] que permite crear documentos con extensas posibilidades de formateo, y la herramienta *sharelatex* [37], que permite la edición de ficheros en LaTex online.

Aspectos relevantes del desarrollo del proyecto

5.1. Explicación general

El TFG consiste en una aplicación que, dadas dos imágenes, una obtenida en el momento de ejecución, ya sea mediante fichero (accediendo a la foto que tengamos almacenada en nuestro equipo), o mediante una captura que realicemos con nuestra cámara, y la otra alojada localmente en una base de datos: sea capaz de distinguir si en ambas imágenes se encuentra la misma persona utilizando técnicas de machine-learning.
A la hora de comparar ambas imágenes, si la persona que se pone delante de la cámara no estaba registrada en la base de datos, aparecerá la persona que esté registrada que más se parezca. En caso de que no se supere un umbral de coincidencia con ninguna de las personas registradas, mostrará una imagen por defecto avisando de que no se ha podido obtener ningún resultado satisfactorio.
En cualquier caso, junto con la imagen que se obtenga como resultado, se mostrará una barra de progreso, que indica el porcentaje de acierto que ha obtenido al encontrar dicho resultado. Además, se muestra un botón que nos permite crear otra ventana extra con información de la persona obtenida como resultado (nombre, edad, lugar de nacimiento y profesión).

5.2. Requisitos técnicos

Es necesario tener instalado *python 2.7* y diferentes librerías: *cv2* (machine-learning orientado a imágenes), *tKinter* (interfaz), *numpy* (utilidad), *Pillow* (operaciones con imágenes). Todas ellas se pueden encontrar en la sección 3.3.

5.3. Fases de desarrollo

Las fases que se han seguido a la hora de desarrollar el proyecto han sido [38]:

- **Definir el proyecto**

  Al principio se tuvo que escoger el tipo de proyecto que se iba a realizar. En nuestro caso un proyecto medio, ya que no se tenían ni el tiempo ni los recursos necesarios para que fuera un proyecto demasiado grande, y tampoco podía ser un proyecto demasiado pequeño dado el objetivo del mismo.

  También se tuvo que definir la idea principal, que en nuestro caso iba a ser reconocer gente.

- **Identificar información, recursos y requisitos**

  A continuación, se definió qué tipo de información iba a tratar: íbamos a trabajar con imágenes en sus diferentes formatos, y con redes neuronales, las cuales íbamos a entrenar para que reconocieran a la gente almacenada en nuestra base de datos.

  Los recursos de los que íbamos a disponer eran:

  1. La imagen que obtuviéramos con la cámara.
  2. Otras imágenes de gente a la que necesitáramos reconocer (en nuestro caso gente desaparecida o en busca y captura). Dichas imágenes se obtendrían de una base de datos oficial de la policía/gobierno.
  3. Información básica sobre cada una de las personas que tuviéramos almacenadas en la base de datos de nuestro programa.

  En cuanto a los requisitos que necesitábamos cumplir, se establecieron los siguientes:

  1. Obtener imágenes en tiempo real.
  2. Entrenar una red neuronal con machine-learning o derivados (deep learning, computer vision, etc).
  3. Tener una base de datos con imágenes que usaríamos para entrenar dicha red.
  4. Realizar una comparación satisfactoria de la persona identificada en la imagen en tiempo real.

5. Mostrar resultados e información en una interfaz al usuario.

- **Fase de planificación**

  Cuando se empezó a desarrollar el proyecto, se sabía cual era el objetivo general del proyecto, aunque no el camino exacto que iba a seguir, de manera que se optó por seguir una serie de hitos principales, enumerados en el apartado anterior, y a partir de esos hitos ir creando otros más pequeños y asequibles a la hora del desarrollo.

  No se pretendía ir completando los hitos uno por uno y que no se pudiera empezar a desarrollar uno hasta que se acabara el anterior, de manera que el progreso se ha ido realizando sobre todos ellos de manera más o menos equivalente.

  Los hitos grandes no tenían fecha prevista de ser completados, sino que se completarían cuando no quedaran tareas pendientes relacionadas con dicho hito, de manera que era complicado planificar el desarrollo para que coincidiera con unos plazos concretos, así que se planificaron el resto de tareas más pequeñas, completando varias para cada reunión.

  Dichas reuniones se llevaban a cabo en función del trabajo planteado de una a otra, así que no eran todas las semanas, sino que se adaptaban un poco al trabajo pendiente, aunque si que es cierto que se han tenido reuniones cada (como máximo) dos semanas, para llevar un seguimiento más o menos regular del desarrollo y que no se quede el mismo estancado en el mismo punto demasiado tiempo.

  En las herramientas que se han barajado utilizar respecto a la planificación, se barajó utilizar *Trello* [4], aunque se descartó ya que, sin otros miembros que aporten contenido, no tenía sentido tener una lista de tareas 'to do' o 'doing' para una sola persona.

  También se pensó en utilizar *Zenhub* [52] para la planificación de hitos, tareas y sprints, aunque se descartó la idea al no tener unos plazos fijos de entrega para cada una de ellas.

  En la tabla 5.1 se pueden observar tanto los milestones principales del desarrollo del proyecto como las issues o tareas asociadas a cada uno.

- **Fase de investigación**

  A la vez que se realizaba la planificación de las tareas semanales, se iba realizando un proceso de investigación e información, tanto de conceptos y mecánicas como de seguimiento de tutoriales, documentación y diferentes referencias sobre los temas tratados en el proyecto (para más información consultar el apartado 2.3).

- **Desarrollo**

  En este apartado hemos ido realizando el desarrollo del propio proyecto, tanto la estructura de la fase de planificación como el propio código. Para ello se ha seguido un orden lógico de desarrollo, centrándonos primero en la funcionalidad básica de tomar una imagen de la cámara y compararla con las de la base de datos, para pasar más tarde a ampliar el número de muestras de nuestra red para mejorar el ratio de acierto de las predicciones, crear un interfaz gráfico para mostrar los resultados y que al usuario le resulte más sencilla su interpretación y crear un ejecutable para evitar tener que ejecutar nuestro código completo en un editor.

  Estas últimas modificaciones se toman para facilidad del usuario, intentando automatizar todo lo posible el proceso, aunque es cierto que se necesitan ciertos datos por parte del usuario, además de permitirle cierta libertad (por ejemplo, podríamos entrenar la red a cada ejecución, incluso si nuestras muestras no han variado y la red está correctamrnte entrenada, pero supondría una pérdida de tiempo y recursos en muchos casos, de manera que se opta por darle opción al usuario).

- **Revisión y control continuo**

  Además de las reuniones (semanales o bisemanales) mencionadas en la planificación, se ha ido llevando un control de errores por cuenta propia, que se iban resolviendo inmediatamente si eran urgentes, o después de realizar las tareas asignadas para esa semana si no afectaban al uso general del programa.

  Con el proyecto en *Github*, ha sido sencillo seguir el progreso y desarrollo del mismo, sabiendo en todo momento qué tareas estábamos realizando, cuales nos quedaban por realizar, cuales eran los objetivos

generales, etc.

Milestones	Issues	Tipo de tarea
1.- Install and Configure	Install environment	Develop
	Create class diagram	Develop
	Basic readme	Develop
2.- Image manipulation	Open image from file	Develop
	Save images	Develop
	Tensorflow Tutorials	Develop
	Meeting	Meeting
3.- Basic image recognition	Import OpenCV	Develop
	Capture image from camera	Develop
	Normalize illumination	Enhancement
	Create basic parameters	Develop
	Facial recognition (I)	Develop
	Meeting	Meeting
4.- Apply computer vision	Improve camera recording	Enhancement
	Change face location display	Enhancement
	Facial recognition (II)	Develop
	Meeting	Meeting
	Acquire camera	Enhancement
5.- Create database	Create local storage	Develop
	Get more images	Enhancement
6.- Interface	Basic interface	Develop
	Dynamic window	Enhancement
	Facial recognition (III)	Enhancement
	Camera view	Enhancement
	Result images	Develop
	Compare bar/percentage	Develop
	Meeting	Meeting
	Result information	Develop
	Executable	Develop
7.- Report	Basic report	Develop
	Meeting	Meeting
	Clean and order code	Enhancement
	Update readme	Enhancement
	Final report	Enhancement
	Video	Develop

Tabla 5.1: Distribución del desarrollo del proyecto según Milestones e Issues.

Trabajos relacionados

Algunos de los proyectos relacionados con el presente que se han consultado han sido:

- *Sistema de Reconocimiento Facial*, por *Germán Matías Scarel* [36].
- *Reconocimiento facial mediante el Análisis de Componentes Principales*, por *Sara Domínguez Pavó* [8].
- *Reconocimiento de imágenes utilizando redes neuronales artificiales*, por *Pedro Pablo García García* [12].
- *Interfaces Hombre Máquina Basados en Hardware Libre*, por *José Ramón Cuevas Diez* [6].
- *GoBees - Monitorización del estado de una colmena mediante la cámara de un smartphone*, por *David Miguel Lozano* [25].
- *Reconocimiento de señales de tráfico mediante Raspberry Pi*, por *Alberto Miguel Tobar* [41].
- *Visión Artificial Aplicada a la Clasificación basada en Color*, por *María Viyuela Fernández* [42].

Conclusiones y Líneas de trabajo futuras

7.1. Base de datos

Ahora mismo hay una cantidad de imagenes en la base de datos suficiente para testear y probar el funcionamiento del programa a nuestro nivel de desarrollo, pero eso en un futuro habría que ampliarlo, dependiendo de la gente que queramos reconocer. No se han querido meter demasiadas imagenes para no tener una carpeta que ocupe demasiado solo con imagenes de gente. Por supuesto, para una aplicación real tendría que tener tantas como se tengan almacenadas en el registro correspondiente.

El formato de dichas imagenes es muy irregular, cada una está tomada desde un ángulo diferente, con una calidad y tamaño variables, debido a que están tomadas de internet, y a partir de estas fotos aleatorias de internet se han guardado en otra carpeta la parte exclusiva del rostro, descartando el resto de la imagen, haciendo que todas tengan el mismo tamaño y una calidad (salvando las distancias) similar. Por supuesto, van a seguir estando cada una tomada desde un ángulo y con una influencia de iluminación y color variable.

Este aspecto, si se tuviera la posibilidad (como en el registro que comentaba antes), lo mejor sería que todas tuvieran el mismo formato (sacadas del registro del dni, registro en instituciones públicas, anteriores condenas, etc), pero no estamos hablando de una aplicación a escala real, sino de un entorno de prueba, en el cual se ha desarrollado el proyecto, por lo que es complicado obtener un standard a partir de imagenes aleatorias obtenidas de internet.

Una posible solución para esto podría haber sido intentar conseguir un foco, una cámara de fotos profesional (o de mayor calidad que una del móvil o una webcam del ordenador), y preguntar a amigos y conocidos que se pusieran ropa similar (camiseta oscura) sobre un fondo claro para obtener los mejores resultados posibles. Esto requeriría de mucho tiempo y recursos (de los cuales no se disponía), y tiene más importancia a la hora de poner en marcha de manera más comercial el proyecto.

7.2. Entrenamiento

En el estado de desarrollo en el que se encuentra el proyecto, cada vez que se introduzca una imagen nueva, se tiene que entrenar la red, en caso contrario se trabajará sobre las imagenes de la base de datos sin actualizar, de manera que se pueden dar lecturas fantasma o falsas identificaciones.

Esto se debe a que, cuando se entrena la red, lo que se hace realmente es extraer las características del rostro de cada una de las imagenes y se guardan en un fichero, por lo que si alguna imagen cambia o se añaden o eliminan es necesario repetir el proceso para recoger los nuevos datos.

Esto podría adaptarse si se quiere conseguir una mejora en el rendimiento cuando se insertan imagenes a menudo, poniendo las funcionalidades de entrenamiento de la red en un programa separado del principal, de manera que, al insertar una nueva imagen, se ejecute solamente ese programa, y al ejecutar el programa principal ya se ha entrenado la red previamente, por lo que tiene que hacer simplemente es el reconocimiento propiamente dicho, sin perder tiempo ni recursos en entrenar y extraer las características.

Otra opción sería extraer las características de cada imagen por separado, de manera que no se debería entrenar la red con todas las imagenes cada vez que se introduce una, sino entrenar la red para la nueva imagen. El problema con este último método, es que necesitaríamos "desentrenar"la red para las imagenes que eliminaramos, de manera que tendríamos que llevar un registro de las imagenes antiguas y de las nuevas, algo que, en una fase inicial del proyecto se descartó, ya que implicaría desarrollar una base de datos compleja, para la que no se tenía tiempo.

Un ejemplo del problema anterior es el siguiente:

Si hay una imagen que se llama 'persona1.png' para una persona [1], se crea el fichero 'entrenamiento-persona1.yml', después de lo cual se borra la imagen y se introduce otra imagen, de otra persona diferente (persona [2]), con el mismo nombre de fichero 'persona1.png'. Al ir a comprobar el control de versiones mencionado en el apartado anterior, nos va a decir que si que se ha entrenado ya esa imagen, pero realmente el fichero 'entrenamiento-persona1.yml' lo que tiene es el entrenamiento de la persona [1] que ya no está en las imagenes, así que volvería a dar errores y falsos positivos.

Bibliografía

[1] OpenCV Version 2.4.13. Haar feature-based cascade classifier for object detection. https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html, 2018.

[2] OpenCV Version 3.4.1. Face detection using haar cascades. https://docs.opencv.org/3.4/d7/d8b/tutorial_py_face_detection.html, 2018.

[3] Inc Anaconda. Anaconda. https://anaconda.org/anaconda/python, 2018.

[4] Atlassian. Trello. https://trello.com.

[5] Alex Clark and Contributors. Pillow. https://pillow.readthedocs.io/en/5.1.x/, 2018.

[6] José Ramón Cuevas Diez. *Interfaces Hombre Máquina Basados en Hardware Libre*. Universidad de Burgos, Burgos, España, 2017.

[7] Google Developers. Image recognition. https://www.tensorflow.org/tutorials/image_recognition, 2018.

[8] Sara Domínguez Pavón. *Reconocimiento facial mediante el Análisis de Componentes Principales*. Universidad de Sevilla, Sevilla, España, 2017. http://bibing.us.es/proyectos/abreproy/91426/fichero/TFG_SARA_DOMINGUEZ_PAVON.pdf.

[9] Jan Fajfr. The basics of face recognition. https://blog.octo.com/en/basics-face-recognition, 2011.

[10] Python Software Foundation. Python - version 2.7. https://www.python.org/downloads/release/python-2715, 2018.

[11] Python Software Foundation. Tkinter — python interface to tcl/tk. https://docs.python.org/2/library/tkinter.html, 2018.

[12] Pedro Pablo García García. *Reconocimiento de imágenes utilizando redes neuronales artificiales*. Universidad Complutense de Madrid, Madrid, España, 2013. http://eprints.ucm.es/23444/1/ProyectoFinMasterPedroPablo.pdf.

[13] Adam Geitgey. Machine learning is fun! part 4: Modern face recognition with deep learning. https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learni 2016.

[14] Adam Geitgey. Machine learning is fun! part 4: Modern face recognition with deep learning. https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learni 2016.

[15] Github. Github. https://github.com/, 2018.

[16] Github. Github desktop. https://desktop.github.com/, 2018.

[17] Google. Google cloud vision api documentation. https://cloud.google.com/vision/docs.

[18] Bharath Hariharan. How are eigenvectors and eigenvalues used in image processing? https://www.quora.com/How-are-Eigenvectors-and-Eigenvalues-used-in-image-processing, 2013.

[19] Stack Exchange Inc. Stackoverflow. https://stackoverflow.com, 2018.

[20] JetBrains. Pycharm. https://www.jetbrains.com/pycharm, 2018.

[21] Andrej Karpathy Justin Johnson. Convolutional neural networks for visual recognition. http://cs231n.github.io/convolutional-networks, 2018.

[22] kaboomfox. Overlay a smaller image on a larger image python opencv. https://stackoverflow.com/questions/14063070/overlay-a-smaller-image-on-a-larger-image-python-opencv, 2012.

[23] KP Kaiser. Deal with it in python with face detection. https://dev.to/burningion/deal-with-it-in-python-with-face-detection-chi, 2017.

[24] Antonio Leiva. Patrones de diseño de software. https://devexperto.com/patrones-de-diseno-software/, 2016.

[25] David Miguel Lozano. *GoBees - Monitorización del estado de una colmena mediante la cámara de un smartphone.* Universidad de Burgos, Burgos, España, 2017.

[26] JGraph Ltd. Draw-io. https://www.draw.io, 2018.

[27] A. Martinez. Fisherfaces. *Scholarpedia*, 6(2):4282, 2011. revision #91266.

[28] Ltd MKLab Co. Staruml. http://staruml.io, 2018.

[29] OpenCV. haarcascades. https://github.com/opencv/opencv/blob/master/data/haarcascades, 2018.

[30] OpenCV. Opencv - releases. https://opencv.org/releases.html, 2018.

[31] Christian S. Perone. Deep learning – convolutional neural networks and feature extraction with python. http://blog.christianperone.com/2015/08/convolutional-neural-networks-and-feature-extraction-with-python, 2015.

[32] Margaret Rouse. Inteligencia artificial, o ai. https://searchdatacenter.techtarget.com/es/definicion/Inteligencia-artificial-o-AI, 2017.

[33] Cesar Troya S. Lbp y ulbp – local binary patterns y uniform local binary patterns. https://cesartroyasherdek.wordpress.com/2016/02/26/deteccion-de-objetos-vi/, 2016.

[34] ANKIT SACHAN. Convolutional neural network (cnn). http://cv-tricks.com/tensorflow-tutorial/training-convolutional-neural-network-for-image-classificatio, 2017.

[35] Kelvin Salton. Face recognition: Understanding lbph algorithm. https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b, 2017.

[36] Germán Matías Scarel. *Sistema de Reconocimiento Facial*. Universidad Nacional del Litoral, Santa Fe, Argentina, 2010. http://pdi-fich.wdfiles.com/local--files/investigacion/PF_Scarel_SistemaReconocimientoFacial.pdf.

[37] ShareLaTeX. Sharelatex. https://es.sharelatex.com, 2018.

[38] sinnaps. MetodologÍa de un proyecto. https://www.sinnaps.com/blog-gestion-proyectos/metodologia-de-un-proyecto, 2018.

[39] Princeton University Stanford Vision Lab, Stanford University. Imagenet. http://www.image-net.org/index, 2016.

[40] Tensorflow. Tensorflow. https://www.tensorflow.org, 2018.

[41] Alberto Miguel Tobar. *Reconocimiento de señales de tráfico mediante Raspberry Pi*. Universidad de Burgos, Burgos, España, 2017.

[42] María Viyuela Fernández. *Visión Artificial Aplicada a la Clasificación basada en Color*. Universidad de Burgos, Burgos, España, 2016.

[43] Chris Welch. Google just gave a stunning demo of assistant making an actual phone call. https://www.theverge.com/2018/5/8/17332070/google-assistant-makes-phone-call-demo-duplex-io-2018, 2018.

[44] Wikipedia. Computer vision — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].

[45] Wikipedia. Desarrollo en cascada — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].

[46] Wikipedia. Latex — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].

[47] Wikipedia. Multilayer perceptron — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].

[48] Wikipedia. Red neuronal artificial — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].

[49] Wikipedia. Redes neuronales convolucionales — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].

[50] Wikipedia. Scrum (desarrollo de software) — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].

[51] Wikipedia. Vector propio y valor propio — wikipedia, la enciclopedia libre, 2018. [Internet; descargado 30-mayo-2018].

[52] ZenHub. Zenhub. https://www.zenhub.com.

[53] S. Zhang and M. Turk. Eigenfaces. *Scholarpedia*, 3(9):4244, 2008. revision #128015.

Resumen

Las tecnologías avanzan continuamente, al igual que hacen las amenazas que acompañan a dichas tecnologías. Por ello, y para mantener una sociedad estable, la seguridad debe ir ligada a estos avances.

En este proyecto se pretende diseñar un sistema de **identificación de personas** a través de un programa de reconocimiento facial, desarrollado en *Python* y utilizando técnicas de **Machine Learning**.

Con este proyecto se pretende aumentar la seguridad en lugares públicos (como pueden ser aeropuertos o centros comerciales) frente a ataques terroristas o cualquier amenaza similar. Para ello, se parte de una base de datos que contiene imágenes de gente en una lista de **busca y captura**. A partir de estas imágenes se pretende identificar a las personas de la base de datos que se encuentren en dichos lugares públicos.

Descriptores

python, inteligencia artificial, IA, reconocimiento facial, cámara, tiempo real, identificación, aprendizaje automático, aprendizaje profundo, entrenamiendo de redes, visión artificial

Abstract

Technology is continuously advancing, so do the threads attached with this technology. For that, and to sustain a stable society, security must suit those advances.

The main goal of this project is to design a **text identifier** system through a facial recognition program, developed in *Python* and using *Machine Learning* technologies.

With this project, we pretend to raise security levels in public places (such as airports or shopping centers) against terrorist attacks or any similar thread. For that, we start with a database with images of people in a *search and capture* list. With those images we can identify people from the database on public places.

Keywords

python, artifical intelligence, AI, facial recognition, camera, real time, identification, machine learning, deep learning, network training, computer vision