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 — 17 de mayo de 2018
Tutor: César Represa

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. César Represa, 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, 17 de mayo de 2018

Vº. Bº. del Tutor:

D. César Represa

Resumen

Se pretende diseñar un sistema de **identificación de personas** a través de un programa de **reconocimiento facial** con rostros previamente incluidos en una base de datos. El sistema de desarrollará en **Python** y utilizando técnicas de **Machine Learning**.

Descriptores

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

Abstract

The main goal is to develop a **people identification** system with a **facial recognition** program and faces included on a database. This system will be developed in **Python** using **Machine Learning** techniques.

Keywords

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

Índice general

Índice de figuras

Índice de tablas

Introducción

El proyecto consiste en una aplicación que, dadas dos imágenes (una obtenida desde una cámara y la otra alojada localmente), sea capaz de distinguir si en ambas imágenes se encuentra la misma persona utilizando técnicas de machine-learning.

El repositorio del proyecto se puede encontrar en el siguiente repositorio.

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

- 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 **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 **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.

- 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 imágen, recortando dicha imágen 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 imágen. Dichos ficheros se proporcionan gratuitamente con fines educativos en el siguiente enlace.

- Carpeta **recognizer**
  Contiene ficheros que utilizamos durante la ejecución del programa para visualizar datos sobre la persona reocnocida o cargar la imágen 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 imágen 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 imágen es la que carga el programa por defecto cuando no se ha conseguido superar el umbral de reconocimiento de un rostro en una imágen. 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 imágen, el programa cargará esta imágen 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 imágen 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 imágen 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 imágen 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 imágen 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 **README.txt**

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

Objetivos del proyecto

El proyecto pretende servir como forma de identificación de personas que, teniendo una imágen 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.

Conceptos teóricos

3.1. Conceptos

Algunos de los conceptos que vamos a necesitar comprender para entender el proyecto son los siguientes:

- **Inteligencia Artificial**

  La inteligencia artificial (IA) según searchdatacenter, 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 enlace se puede encontrar un ejemplo de una red convolucional desarrollada en python.

- **Red neuronal artificial**

  Las **redes neuronales artificales** 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 siguiente imágen.

  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 son un subtipo de las redes neuronales artificiales, que se basan en las versiones biológicas de los perceptrones multicapa (Multi Layer Perceptron en inglés), 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 imágen. Para obtener más información sobre este tipo de redes se pude consultar el siguiente enlace.

- **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.

- **Visión Artificial**

  La visión artificial (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.

- **Reconocimiento Facial**

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

  Este facial recognition tiene 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 los usos en reconocimiento y seguimiento de clientes por parte de empresas y tiendas, así como el reciente "desbloqueo facial"de los teléfonos inteligentes.

  En el siguiente enlace se pueden ver distintas aplicaciones desarrolladas por tensorflow, no sólo para el reconocimiento facial, sino para el reconocimiento de imágenes en general.

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

  - **Detección de rostros**

    En esta primera parte del proceso se **identifican los pixeles de la imágen 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". Todas las versiones de este algoritmo se pueden encontrar en el

siguiente repositorio.

Hay muchas aplicaciones 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 imágen** (iluminación, posición de la persona, cambios en los rasgos faciales, etc) como pueden ser las funciones que realizan cortes o "thresholds"de nuestra imágen a partir de ciertos valores umbrales, y se obtiene como resultado una segunda **imágen más .esandarizada", la cual comparamos** (usando en nuestro caso los CascadeClassifier) 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 imágen (o la que más se acerque).

- **Eigenvectores**

  Un eigenvector o valor propio de una imágen es, según wikipedia, 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 imágen, obtenemos todos sus píxeles, y hacemos que cada uno de ellos represente un elemento de una matriz de tamaño MxN (tamaño de la imágen). **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 imágen**.

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

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

Se pueden obtener más aplicaciones de los eigenvectores relacionados con imágenes en la siguiente página.

- **Algoritmos de Reconocimiento Facial**

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

  - **Eigenfaces**

    Las eigencaras o eigenfaces 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 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, el algoritmo LBP (Local Binary Pattern) es un sistema simple pero eficaz, que consiste en etiqutar los píxeles de una imágen 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. Para ello es necesario tratar la imágen 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 imágen, 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 imágen, 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 siguiente imágen.

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.

Técnicas y herramientas

Se ha utilizado la aplicación image.net para la obtención de imágenes de muestra.

Aspectos relevantes del desarrollo del proyecto

Trabajos relacionados

Sistema de Reconocimiento Facial

Conclusiones y Líneas de trabajo futuras

Referencias bibliográficas

1. Inteligencia artificial
   Author = Margaret Rouse
   Title = Inteligencia artificial, o AI
   Year = 2017
   Link = link

2. Red neuronal
   Author = Wikipedia
   Title = Red neuronal artificial
   Year = 2018
   Link = link

   Author = Wikipedia
   Title = Multilayer perceptron
   Year = 2018
   Link = link

3. Red neuronal convulacional
   Author = Wikipedia
   Title = Redes neuronales convolucionales
   Year = 2018
   Link = link

   Author = cv-tricks
   Title = Convolutional Neural network (CNN)

Link = link

4. Reconocimiento facial
   Author = Jan Fajfr
   Title = The basics of face recognition
   Year = 2011
   Link = link

   Author = Adam Geitgey
   Title = Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning
   Year = 2016
   Link = link

5. Visión artificial
   Author = Wikipedia
   Title = Computer vision
   Year = 2017
   Link = link

6. Computer vision de Google
   Author = Google
   Title = Google Cloud Vision API Documentation
   Link = link

7. Machine Learning is Fun!
   Author = Adam Geitgey
   Title = Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning
   Year = 2016
   Link = link

8. Haar cascade face detection
   Author = OpenCV
   Title = Face Detection using Haar Cascades
   Year = 2018
   Link = link

9. Cascade classifier
Author = OpenCV
Title = Haar Feature-based Cascade Classifier for Object Detection
Year = 2018
Link = link

10. Aplicación de detección de rostros
Author = KP Kaiser
Title = DEAL WITH IT in Python with Face Detection
Year = 2017
Link = link

Author = kaboomfox
Title = Overlay a smaller image on a larger image python OpenCv
Year = 2012
Link = link

11. Eigenfaces
Author = Sheng Zhang and Matthew Turk
Title = Eigenfaces
Year = 2008
Link = link

12. Fisherfaces
Author = Aleix Martinez
Title = Fisherfaces
Year = 2011
Link = link

13. Algoritmo LBPH
Author = Kelvin Salton
Title = Face Recognition: Understanding LBPH Algorithm
Year = 2017
Link = link

14. Redondeos en python
Author = Danielle B

Title = Rounding In Python — When Arithmetic Isn't Quite Right
Year = 2017
Link = link

15. Buscador de imágenes por etiquetas Author = Stanford Vision Lab, Stanford University, Princeton University
Title = Image.net
Year = 2016
Link = link

- Python se puede obtener en el siguiente enlace.

- Para realizar la instalación del entorno virtual utilizado durante el desarrollo del trabajo se ha utilizado Anaconda.

- Se han obtenido respuestas y soluciones a varios de los problemas encontrados durante la realización del proyecto en stackoverflow.

- Para los diagramas de clases se ha usado la herramienta gratuita starUML.io.

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

- Para la memoria se ha utilizado la herramienta que permite la edición de ficheros en LaTex online sharelatex.com.

  Otros enlaces que resultaron útiles a la hora de desarrollar (y resolver problemas) del proyecto fueron:

- Python

  - How do I install pip on Windows?

- Installing packages with pip
- Installing modules with python
- Using wheels to install python dependences
- Library to solve compatibility problems between python 2.7 and 3.6
- ERROR: problema a la hora de usar input()/raw_input() con las diferentes versiones de python
- Clases y paquetes en python
- ERROR: missing 1 required positional argument (Problema relacionado con el uso de clases y objetos)
- Algunos problema con las llamadas a funciones desde el main y desde otras clases
- Clases abstractas e interfaces en python I
- Clases abstractas e interfaces en python II
- Tratamiento de excepciones en python
- Artículo bastante interesante sobre cómo trabaja python con los redondeos y como solucionar posibles problemas que surgen de ello.
- Operadores ternarios
- Operaciones con listas I
- Operaciones con listas II

- Contador de tiempo en python
- Singleton simple en python
- Operaciones con ficheros I
- Operaciones con ficheros II
- Operaciones con ficheros III
- Operaciones con ficheros IV
- Operaciones con ficheros V

- Tensorflow
  - Página con buenos tutoriales sobre el uso general de Tensorflow
  - Tutorial rápido de instalación de tensorflow
  - ERROR: Tensorflow not found in pip
  - Cómo entrenar un modelo de detección de rostros con Tensorflow
  - Usando Tensorflow con la GPU en una red convolucional

- Tratamiento de imágenes
  - Pillow para pycharm
  - DLib para reoconocimiento básico de rostros
  - Cargar, modificar y guardar imágenes (con la librería de cv)
  - Abrir imágenes desde fichero

- OpenCV
  - Open CV
  - Se puede consultar también las siguientes páginas para solucionar alguno de los problemas que pueden surgir de la instalación de OpenCV: I, II, III, IV)
  - Activar cámara
  - Detección de rostros con deep learning (Librería CV en nuestro caso)
  - Cómo funciona realmente el método predict() de open cv.
  - Documentación sobre FaceRecognition de Open CV.
  - ERROR: problema a la hora del tratamiento de typos de las imágenes I
  - ERROR: problema a la hora del tratamiento de typos de las imágenes II
  - Pausar ejecución para ver resultados I
  - Pausar ejecución para ver resultados II
  - ERROR: la cámara muestra imágen en negro - ajustar parámetros
  - Cambiar tamaño de las imágenes obtenidas I
  - Cambiar tamaño de las imágenes obtenidas II
  - Recortar imágenes para guardar en la base de datos sólo los rostros
  - Imágen en b/n utilizando funciones de Threshold (para normalizar iluminación y elementos innecesarios)

- Capturar imágenes con la cámara I
- Capturar imágenes con la cámara II
- Capturar imágenes con la cámara III

- Interfaz
  - Resolución de pantalla con python (en nuestro caso con las librerías de tKinter y wx)
  - Redimensionar automáticamente las ventanas de la interfaz en función de la resolución de nuestra pantalla
  - Mostrar múltiples imágenes en una sola ventana I (con la librería de cv - old version)
  - ERROR: mostrar múltiples imágenes en una sola ventana II (con la librería de matplotlib - old version)
  - Redimensionar imágenes de salida (con la librería de cv - old version)
  - Mostrar imágen capturada con la cámara en sentido correcto
  - Sencillo tutorial para aprender a usar tKinter
  - Juntando los resultados de OpenCV con la interfaz de tKinter
  - Personalizando la interfaz con tKinter
  - ERROR: botón para mostrar información adicional no responde
  - ERROR: abrir dos ventanas conjuntas con tKinter I
  - ERROR: abrir dos ventanas conjuntas con tKinter II

- ERROR: abrir dos ventanas conjuntas con tKinter III
- ERROR: abrir dos ventanas conjuntas con tKinter IV
- Barra de progreso (comparación) en tKinter

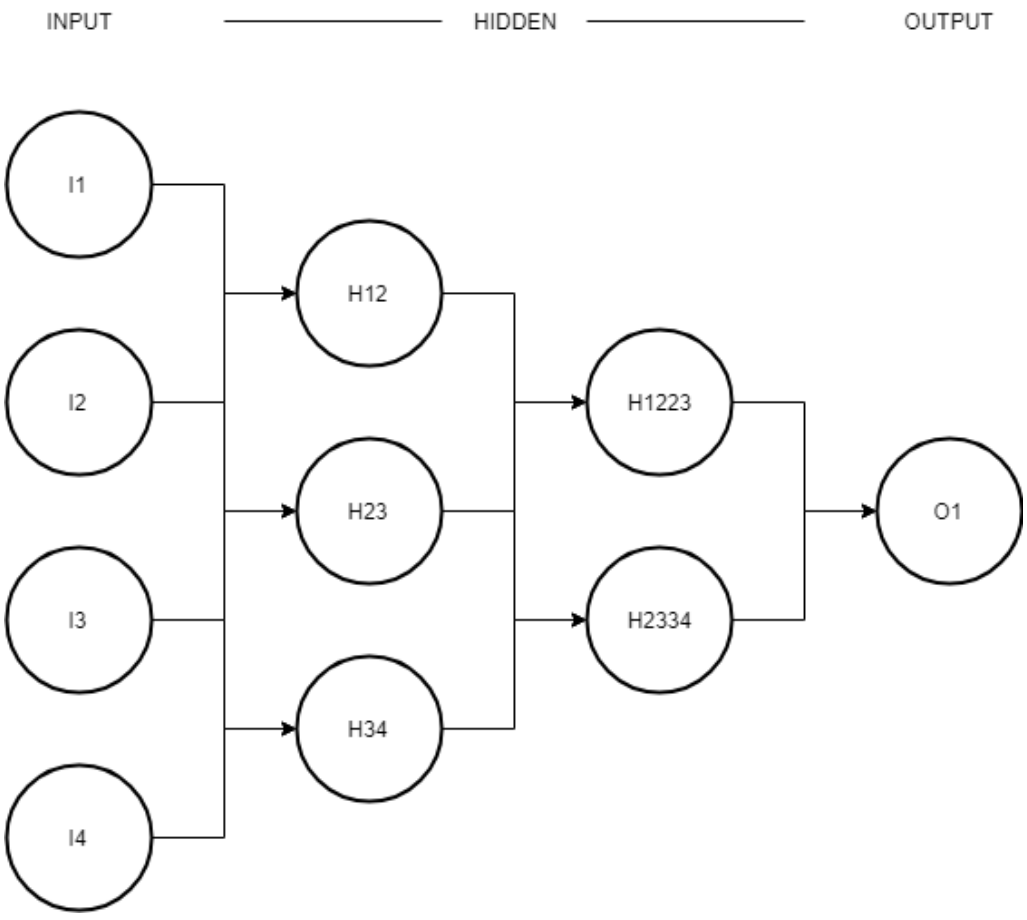
- Otros
  - Eliminar carpeta con contenido en github
  - Base de datos SQLite
  - Matplotlib para msotrar imágenes - old version

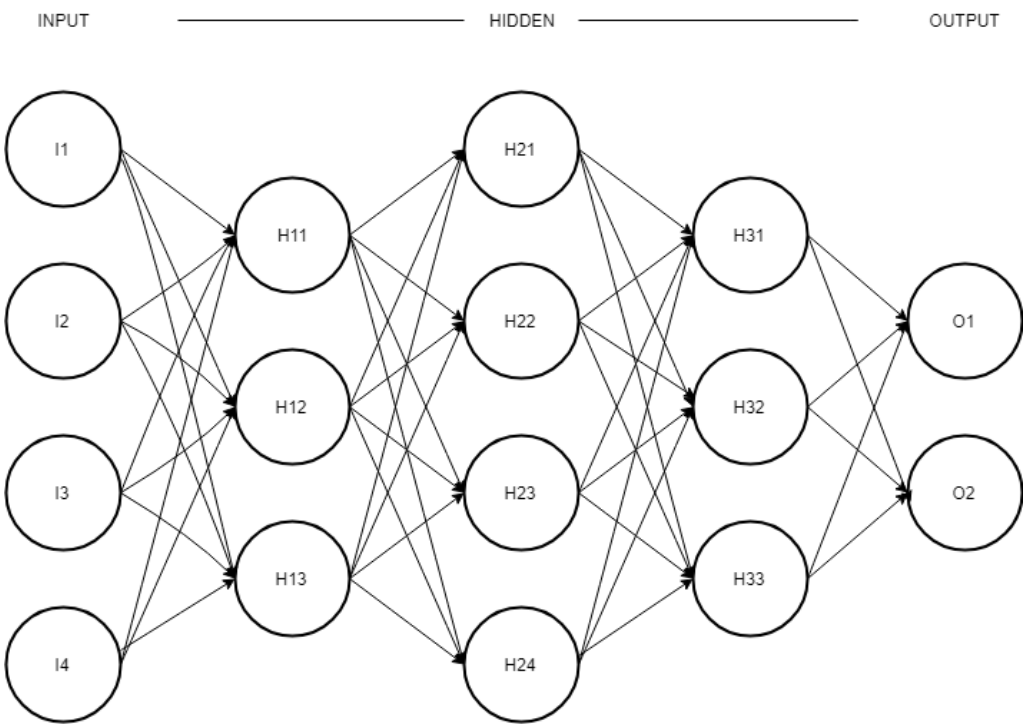
- Repositorios
  - .gitignore
  - Tensorflow
  - Librería de Tensorflow para modelos complejos
  - Clasificación de imágenes usando Tensorflow
  - Detección de rostros con la api de Tensorflow I
  - Detección de rostros con la api de Tensorflow II
  - Google cloud platform: Computer Vision
  - Microsoft Cognitive Toolkit
  - COCO: reconocimiento de objetos a gran escala
  - Facenet: Face recognition using Tensorflow

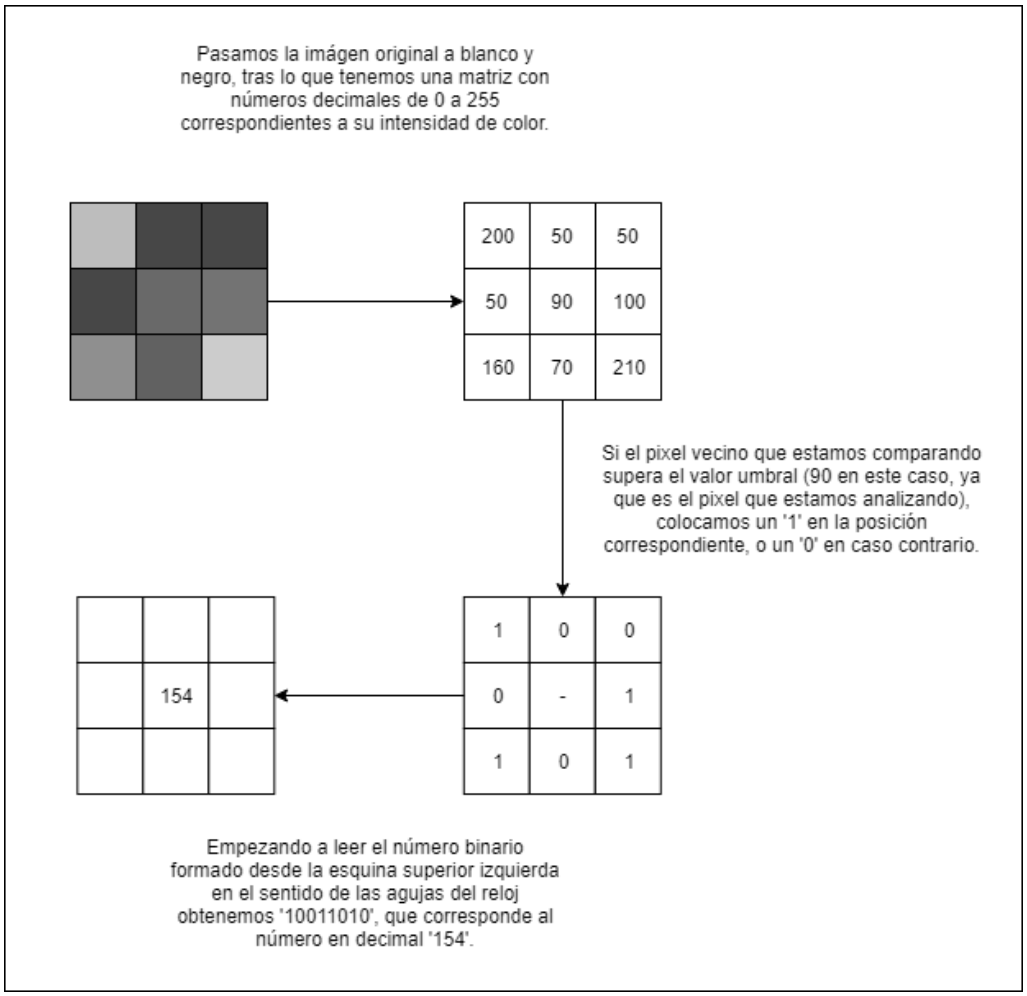
- Face recognition: The world's simplest facial recognition api for Python and the command line

- Red neuronal desarrollada en python con numpy

- Tutoriales sobre computer vision




INPUT
HIDDEN
OUTPUT
I1
I2
I3
I4
H11
H12
H13
H21
H22
H23
H24
H31
H32
H33
O1
O2


Pasamos la imágen original a blanco y negro, tras lo que tenemos una matriz con números decimales de 0 a 255 correspondientes a su intensidad de color.
200 50 50
50 90 100
160 70 210
Si el pixel vecino que estamos comparando supera el valor umbral (90 en este caso, ya que es el pixel que estamos analizando), colocamos un '1' en la posición correspondiente, o un '0' en caso contrario.
1 0 0
0 - 1
1 0 1
154
Empezando a leer el número binario formado desde la esquina superior izquierda en el sentido de las agujas del reloj obtenemos '10011010', que corresponde al número en decimal '154'.