UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

Sistema clasificador de iris

Presentado por Johnson Bolívar Arrobo Acaro
en Universidad de Burgos — 28 de septiembre de 2020
Tutor: José Francisco Diez Pastor

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. José Francisco Diez Pastor, profesor del departamento de Ingeniería Informática, área de Lenguajes y Sistemas informáticos.

Expone:

Que el alumno D. Johnson Bolívar Arrobo Acaro, con DNI 71829434-C, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado "Sistema clasificador de iris".

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, 28 de septiembre de 2020

Vº. Bº. del Tutor:

D. José Francisco Diez Pastor

Resumen

Los sistemas de seguridad basados en la biometría han experimentado una gran demanda y están presentes cada vez más en productos cotidianos como *smartphones* y *tablets*.

En este trabajo se propone un sistema de identificación basado en el iris ocular, ya que se trata de un rasgo que proporciona una precisión muy alta y cuyas características se mantienen estables a lo largo de la vida.

Así mismo, se ahondará en las técnicas existentes para llevar a cabo el proceso de identificación, desde los métodos convencionales usando algoritmos matemáticos hasta los métodos más actuales cómo los algoritmos de *Machine Learninng* y Redes Neuronales.

Descriptores

Biometría, iris, segmentación, sistemas de identificación, Machine Learning, redes neuronales.

Abstract

Biometric-based security systems have been experienced a boom and are increasingly present in everyday products such as *smartphones* and *tablets*.

In this project, an identification system based on the ocular iris is proposed, since it is a feature that provides very high precision and whose characteristics remain stable throughout life.

Likewise, it will investigate into existing techniques to carry out the identification process, from conventional methods using mathematical algorithms to the most current methods such as the use of Machine Learning algorithms and Neural Networks.

Keywords

Biometrics, iris, segmentation, Biometric-based security sustems, Machine Learning, Neural Networks.

Índice general

Índice de figuras

Índice de tablas

Introducción

> La probabilidad de que 2 iris cuenten con el mismo patrón es de aproximadamente 1 entre 10^{78}. (La población mundial es de alrededor de 10^{10})
>
> Frankin Cheung

La biometría se define cómo la toma de medidas estandarizadas de los seres vivos o de procesos biológicos [29] que pueden ser usadas cómo medio para la identificación automática de individuos. Una buena biometría deberá ser:

- **Universal:** cada individuo deberá tener unos determinados rasgos biométricos únicos.
- **Recolectable:** los rasgos biométricos podrán ser medidos y guardados.
- **Permanente:** no deberían variar a lo largo de la vida del individuo.

Los sistemas de reconocimiento basados en la biometría han experimentado un auge muy pronunciado en los últimos años, esto se puede ver reflejado en su implementación en dispositivos cotidianos del día a día, como *smartphones* y *tablets*.

Pero más allá de las huellas dactilares, la geometría de la mano o el reconocimiento facial (entre otros), existe otro rasgo físico nato que está cobrando mucha importancia, ya que comparado con los mencionados anteriormente, es mucho más seguro, se trata del iris ocular.

La idea de usar el iris para la identificación fue propuesto por el oftalmólogo Frank Burch en 1936, pero no fue hasta 1987 cuando Leonard Flon y Aran Safir patentaron la idea de Burch, aunque fueron incapaces de desarrollar por sí mismos los algoritmos necesarios, así que decidieron acudir a John Daugman, profesor en ese entonces de la Universidad de Harvard. Los algoritmos desarrollados por Daugmann en 1994 son la base de todos los sistemas de reconocimiento de iris actuales [11].

El iris cómo rasgo biométrico aporta las siguientes ventajas [7]:

- Es un rasgo que permanece invariable a lo largo de toda la vida del individuo y rara vez se ve afectado a causa de factores externos cómo accidentes o cirugías. Ver Figura 1.1
- Está constantemente protegido por la córnea
- Se trata de una técnica no invasiva, ya que no hace falta el contacto con el individuo para la toma de muestras, y por tanto su aceptabilidad por parte de estos es alta.
- Sus patrones son tan únicos que no existen 2 iris iguales, incluso los iris derecho e izquierdo del mismo individuo son distintos.

Figura 1.1: Sharbat Gula, identificada 18 años mediante la aplicación de los algoritmos de Daugman.

En el siguiente proyecto se expondrá el funcionamiento básico de un sistema de reconocimiento mediante el análisis de los patrones únicos del tejido membrano-muscular del iris mediante técnicas de *Machine Learning*.

1.1. Estructura de la memoria

La memoria sigue la siguiente estructura:

- **Introducción:** breve descripción del problema a resolver y la solución propuesta. Estructura de la memoria y listado de materiales adjuntos.
- **Objetivos del proyecto:** exposición de los objetivos que persigue el proyecto.
- **Conceptos teóricos:** breve explicación de los conceptos teóricos clave para la comprensión de la solución propuesta.
- **Técnicas y herramientas:** listado de técnicas metodológicas y herramientas utilizadas para gestión y desarrollo del proyecto.
- **Aspectos relevantes del desarrollo:** exposición de aspectos destacables que tuvieron lugar durante la realización del proyecto.
- **Trabajos relacionados:** estado del arte en el campo del reconocimiento del iris.
- **Conclusiones y líneas de trabajo futuras:** conclusiones obtenidas tras la realización del proyecto y posibilidades de mejora o expansión de la solución aportada.

1.2. Materiales adjuntos

- **Anexos:**
  - **Plan del proyecto software:** planificación temporal y estudio de viabilidad del proyecto.
  - **Especificación de requisitos del software:** se describe la fase de análisis; los objetivos generales, el catálogo de requisitos del sistema y la especificación de requisitos funcionales y no funcionales.
  - **Especificación de diseño:** se describe la fase de diseño; el ámbito del software, el diseño de datos, el diseño procedimental y el diseño arquitectónico.
  - **Manual del programador:** recoge los aspectos más relevantes relacionados con el código fuente (estructura, compilación, instalación, ejecución, pruebas, etc.).
  - **Manual de usuario:** guía de usuario para el correcto manejo de la aplicación.

- **Notebooks de experimentación:** contienen todos los experimentos realizados y el conjunto de datasets empleados para la realización los mismos.
- **Aplicación de escritorio:** demo realizada en *Tkinter* que muestra la funcionalidad del proyecto.

El proyecto, junto con los materiales mencionados están disponibles en el siguiente repositorio de GitHub: https://github.com/jaa0124/iris_classifier

Objetivos del proyecto

2.1. Objetivos generales

- Investigar sobre las técnicas existentes en el ámbito del reconocimiento de individuos mediante el iris.
- Escoger una de las técnicas y profundizar en ella con el fin de obtener un modelo funcional.
- Facilitar la comprensión de los resultados obtenidos a cualquiera con ningún conocimiento del tema que se trata.
- Desarrollar un aplicación gráfica de escritorio que permita probar la funcionalidad estudiada.

2.2. Objetivos técnicos

- Aprender a usar bibliotecas populares de *Machine Learning* como *Scikit-learn* o *Keras*
- Usar modelos preentrenados de *Deep learning* para la extracción de los patrones del iris.
- Desarrollar una aplicación de escritorio mediante *TkInter*.
- Usar GitHub para alojar el proyecto y realizar su seguimiento y Git como sistema de control de versiones.
- Implementar la metodología ágil estudiada en la carrera, Scrum.

2.3. Objetivos personales

- Desarrollar un proyecto fuera de los propuestos por la Universidad.
- Comprender el funcionamiento de un sistema de reconocimiento e intentar realizar alguna aportación personal en dicho campo.
- Aplicar y ampliar los conocimientos adquiridos durante la carrera.
- Adentrarme en el mundo de la inteligencia artificial de manera autodidacta o mediante cursos online como Udemy y Coursera.
- Adentrarme en el desarrollo de aplicaciones de escritorio.

Conceptos teóricos

3.1. Anatomía del iris

El iris se corresponde con la parte coloreada del ojo, se trata de un músculo circular (o elíptico) protegido por la córnea en cuyo centro se encuentra la pupila. Consta de 2 músculos el *dilatador* y el *esfínter* cuya función es ajustar el tamaño del iris para controlar la cantidad de luz que entra en la pupila. El conjunto iris-pupila se encuentra rodeado por una membrana de color blanco llamada *esclera*. Ver Figura 3.2

El color, la textura y los patrones del iris son únicos y se cree que se forman aleatoriamente durante el periodo embrionario (etapa comprendida entre fecundación y la octava semana de embarazo), de modo que incluso 2 gemelos genéticamente iguales tienen distintos patrones.

En un entorno con mucha luz el esfínter contrae la pupila, dejando pasar menos luz a la retina, mientras que en un entorno poco iluminado, el músculo dilatador, dilata la pupila para permitir la entrada de más luz. El ojo está protegido externamente por los párpados y las pestañas, sin embargo suponen un problema ya que pueden ocultar significativamente el iris y por tanto resultar en un mal reconocimiento.

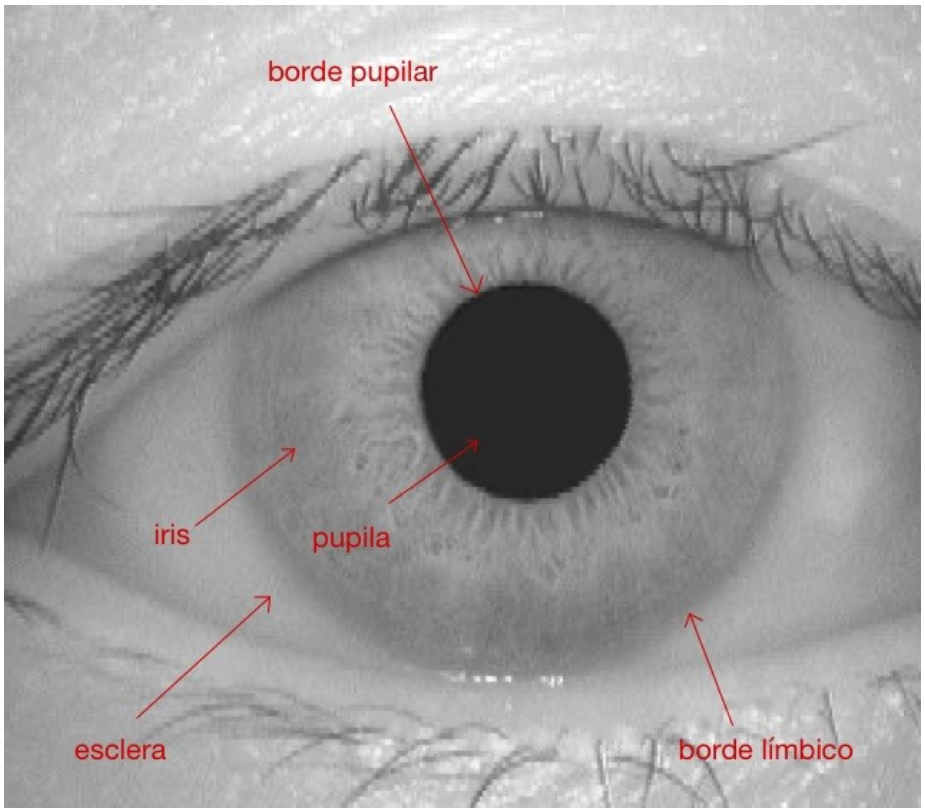

Figura 3.2: Imagen 001_1_1.bmp del dataset CASIA-Iris V1 que muestra las partes del ojo.

3.2. Machine Learning

El *Machine Learning* es la rama de la Inteligencia Artificial que se encarga del desarrollo de técnicas que permitan extraer por sí mismos los patrones y relaciones existentes en los datos, es decir, su objetivo es conseguir que las máquinas aprendan sin necesidad de intervención humana [32, 27].

A partir de unos datos de entrada (*input/X*), y un resultado esperado para esos datos(*outputs/y*), un algoritmo de *Machine Learning* analizará las posibles relaciones entre estos y aprenderá de ellas (se entrenará) para realizar predicciones futuras.

Tipos de Machine Learning

- **Aprendizaje supervisado** este tipo de algoritmos generan una función que establece una relación entre los datos de entrada y las salidas deseadas (etiquetas/*labels*). Existen 2 tipos:
  - **Clasificadores:** clasifican los datos de entrada en grupos o clases. Existen 2 tipos de problemas de clasificación:
    - **Binario:** existen únicamente 2 clases (0/1, *true*/*false*, o positivo/negativo)
    - **Multiclase:** existen 3 o más clases.
  - **Regresores:** predicen un valor o cantidad.
- **Aprendizaje no supervisado** estos algoritmos sólo disponen de los datos de entrada, por lo que extraerán los patrones de los datos sin tener referencia alguna. Un ejemplo de este tipo de aprendizaje es el agrupamiento o *clustering*
- **Aprendizaje por refuerzo** abarcan aquellos algoritmos que implementa un sistema de recompensas en función de las decisiones tomadas por un agente influenciado por el medio o entorno que le rodea.

En este proyecto se hará uso del *Aprendizaje supervisado*, en el que tendremos como *inputs* los atributos del iris, y cómo etiquetas, los nombres del individuo al que pertenecen dichos atributos. Así mismo se planteará como un problema de clasificación multiclase en el que se cuenta concretamente con 108 clases que se corresponderan con los 108 individuos del dataset de CASIA V1 3.5

Para la última fase del proceso del reconocimiento se usarán 4 clasificadores:

- **Support Vector Machines**
- **Logistic Regression**
- **Nearest Neighbours**
- **Random Forest**

3.3. Deep Learning

Las algoritmos de *Machine Learning* convencionales están limitados por su capacidad para procesar los datos de entrada.

Como alternativa surge el llamado *Deep Learning* el cual permite desarrollar modelos computacionales compuestos internamente por múltiples capas de procesamiento.

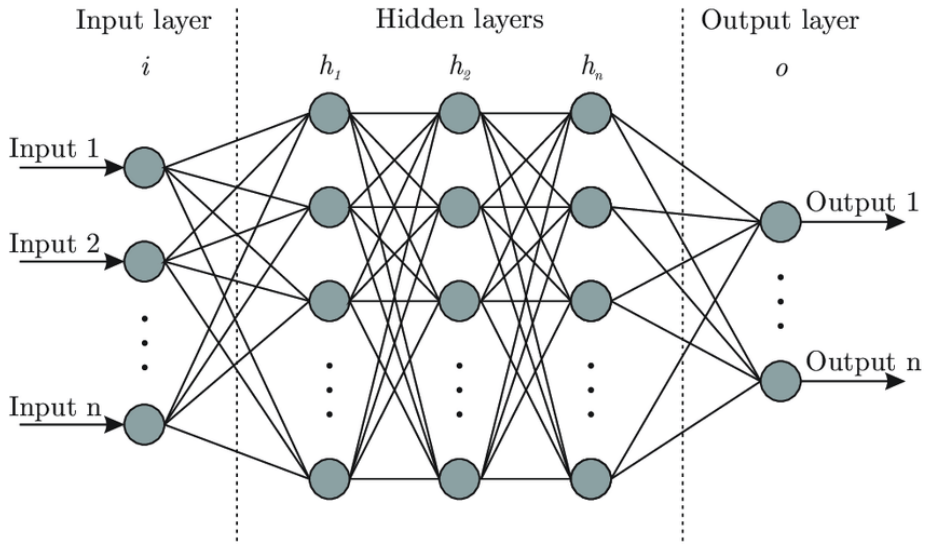

Figura 3.3: Red Neuronal [25]

A la primera capa de la red se la denomina *capa de entrada* y contiene las variables de entrada, le siguen una serie de capas intermedias denominadas *capas ocultas*, donde las neuronas que se encuentren en la misma capa, recibiran la información procesada por la capa anterior y los cálculos que realice en dicha capa los pasará a la siguiente. Finalmente está la *capa de salida* que nos proporcionará el resultado de la red.

3.4. Fases del reconocimiento

Un sistema de reconocimiento tiene 3 fases:

1. Adquisición de imágenes.
2. Preprocesamiento de las imágenes:

   La cual se divide en 3 subfases:

   a) Segmentación.
   b) Normalización.
   c) Extracción de atributos/*features* (patrones del iris)

3. Clasificación de las imágenes.

3.5. Adquisición de imágenes

Es la primera de todas las fases y también la más trascendental ya que es necesario que las muestras tengan la calidad necesaria para que la posterior extracción de patrones sea eficiente.

Las muestras de un iris se toman con cámaras desarrolladas específicamente para este propósito si bien para este proyecto no se contaba con este hardware se podría haber optado por usar una cámara convencional [23], pero ello incluiría otro problema: conseguir muestras de un gran número de voluntarios. Para evitarse los problemas mencionados se optó por usar un dataset de iris, concretamente el de CASIA en su versión 1 [1] el cual está disponible de forma gratuita mediante registro y autorización previa.

Este dataset cuenta con 756 imágenes del iris de 108 sujetos. La toma de muestras se realizó en 2 sesiones, tomándose 3 muestras en la primera sesión y 4 en la segunda, de modo que se cuenta con un total de 7 muestras por sujeto. Cada muestra está en formato *.bmp* y tienen una resolución de 320x280. Para la toma de muestras usaron una cámara desarrollada por la propia CASIA.

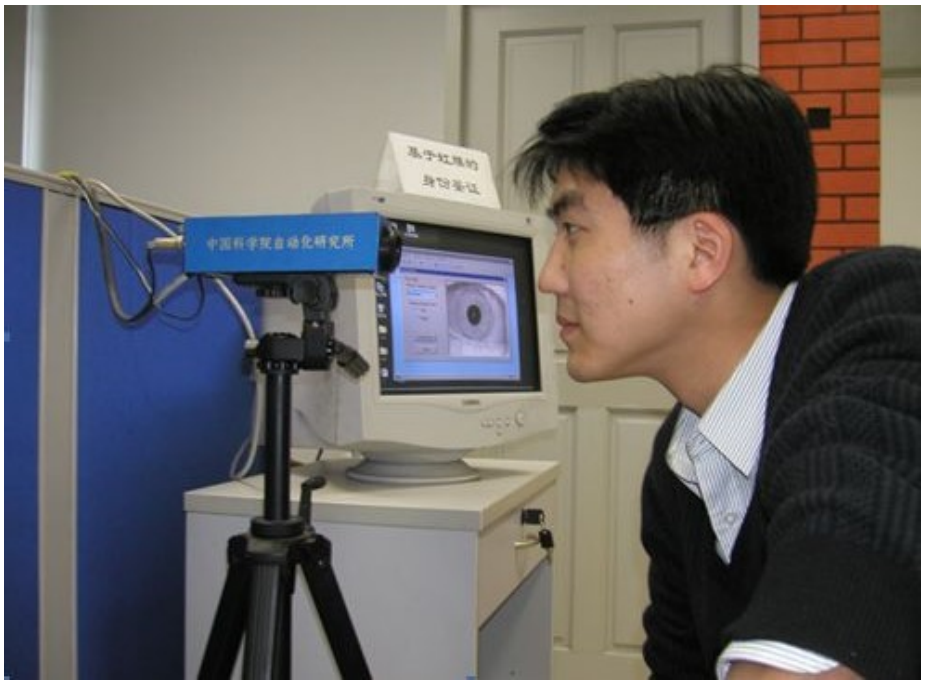

Figura 3.4: Toma de muestras para el dataset CASIA-Iris-V1

Con el objetivo de facilitar la detección de los bordes del iris, las muestras se han editado, de modo que se han eliminado los reflejos especulares [1].

Cabe recalcar que dichas ediciones son mínimas y se aplican únicamente en la pupila [16], la cual no proporciona ninguna información útil para las etapas posteriores de extracción y clasificación de los patrones del iris [8].

[1] Son las regiones más brillantes de la muestra del ojo, es decir, aquellos pixeles con valores iguales a 0. La superficie reflectante de la córnea y la fuente de luz usada para tomar la muestra son la principal causa de la aparición de estos reflejos.

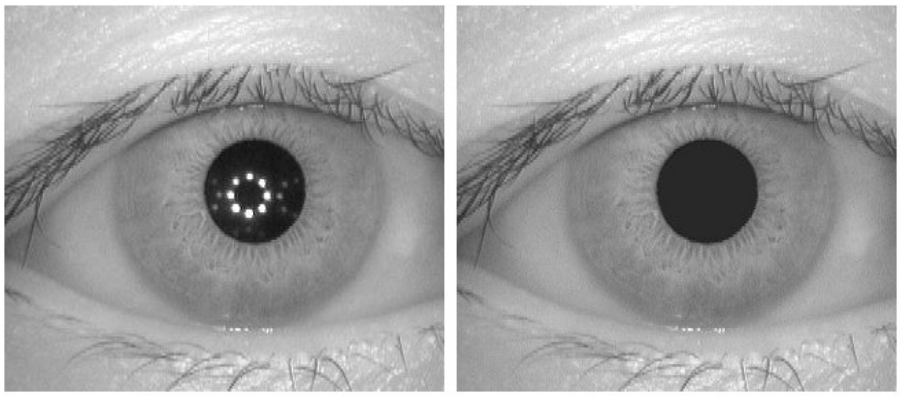

Figura 3.5: Imagen S1143R01.jpg perteneciente CASIA-Interval-V3 con regiones especulares(izquierda). Imagen 001_1_1.bmp de CASIA-Iris-V1 sin regiones especulares(derecha)

3.6. Preprocesamiento

Como puede apreciarse en la figura de la derecha de la imagen 3.5, la muestra es del ojo completo, pero para el reconocimiento únicamnete nos interesa la región del iris, por lo que elementos como pestañas, párpados, esclera *etc* han de eliminarse ya que no aportan nada. Así mismo dependiendo de las condiciones en las que se tomaron las muestras es posible que el tamaño de los iris varíen debido a la dilatación o la contracción de la pupila.

Es por ellos que ha de procesarse la imagen para localizar y aislar el iris (*segmentar*) y ajustarlo a un tamaño estándar (*normalizar*)

Segmentación

En esta etapa se requiere detectar de manera automática los bordes *límbico* y *pupilar* del ojo aislando la región del iris y excluyendo el resto de la imagen, es decir, se busca el *ROI (Region of interest)*

El iris se corresponde con la región circular comprendido entre la pupila y la esclera. La mayoría de fallos en un sistema de reconocimiento de iris se debe a una segmentación deficiente de las imágenes, por lo que se han de tener en cuenta los siguientes factores:

- El iris se encuentra obstruido por pestañas, párpados y reflejos especulares.

- Tanto iris como pupila no son siempre circulares sino que pueden tener forma elíptica, y su forma puede variar dependiendo de la forma en que se tome la muestra.
- Factores cómo muestras desenfocadas, con mucho ruido, poca calidad.. (estos han de tenerse en cuenta en la etapa de adquisición de imágenes)

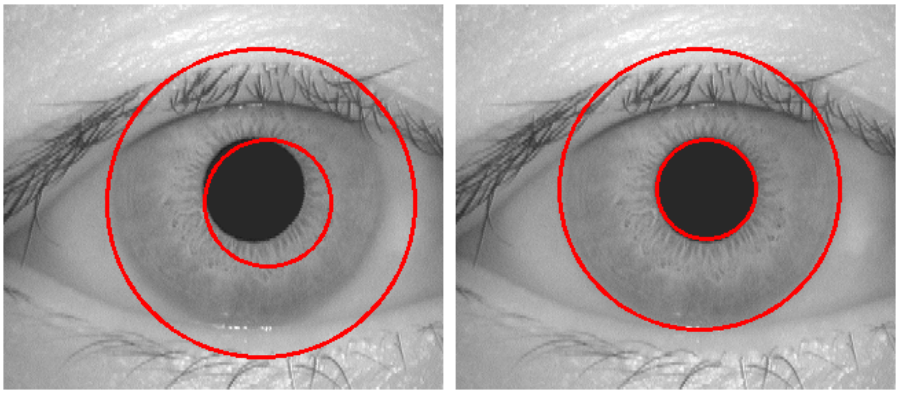

Figura 3.6: Detección incorrecta de los bordes límbico y pupilar (izquierda) y la correcta (derecha)

Como se ha dicho anteriormente, los algoritmos de Daugman son la base de los sistemas de reconocimiento actuales, pero no son los únicos métodos existentes.

- **Operador Integro-Diferencial de Daugman:** Daugman propuso el primer modelo funcional para un sistema de reconocimiento de iris y consiste en un detector de bordes circulares que busca el máximo valor del cambio de tonalidad en la imagen utilizando la integral de línea de varios círculos concéntricos.

  El operador viene determinado por la siguiente ecuación [15]:

  añdir refrencia iris detection and noramlization[4]

$$\max_{(r,x_0,y_0)} \left| G_\sigma(r) * \frac{\partial}{\partial x} \oint_{(r,x_0,y_0)} \frac{I(x,y)}{2\pi r} ds \right|$$

  donde la intendidad del pixel en las coordenadas $I(x, y)$ se corresponde con la imagen del ojo, ds es el arco diferencial del círculo de radio r y coordenadas (x_0, y_0). G_σ es una función que suaviza la imagen

como un filtro Gaussiano de tamaño σ. Dado que utiliza información derivativa de primer orden, se computa muy rápidamente [38].

Lo que hace la ecuación es encontrar el máximo cambio entre las intensidades medias de dos circunferencias concéntricas consecutivas. En el borde entre el iris y la esclera este cambio será máximo porque la última circunferencia concéntrica del iris será oscura y la primera de la esclera será blanca. Dicho cambio viene dado con la derivada.

La media de todos los píxeles de cada una de las posibles circunferencias concéntricas viene de la integral de la división entre la intensidad y $2\pi r$ ya que esa es la longitud de la circunferencia.

- **Transformada de Hough:** parte de la idea de que los bordes límbico y pupilar pueden ser considerados como círculos, la Transformada de Hough original se usaba en un principio para la detección de líneas en una imagen, sin embargo más adelante se extendió para que pudiera reconocer también circulos y circunferencias, pasó a denominarse *Transformada de Hough Circular*.

  Su funcionamiento es el siguiente: Primero se realzan los bordes para generar el *edge map* de la imagen mediante el cálculo de la primera derivada de las intensidades de los píxeles y a continuación se establece un valor que permita anotar como bordes los cambios de intensidad superiores a dicho valor (umbral/*thresholding*). Posteriormente se suaviza la muestra con un filtro Gaussiano.

- **Detector de bordes de Canny:** se basa en detección de contornos mediante el uso de máscaras de convolución y el cálculo de la primera derivada. Los puntos que conforman los bordes detectados se consideran zonas de píxeles en los que existe un cambio brusco del nivel de gris [28]

El método seguido para la realización de este proyecto se basa en el propuesto por Wildes [13]

Wildes propone utilizar el detector de bordes de Canny seguido de la Transformada de Hough Circular, la combinación de ambos métodos proporciona resultados que pueden superar a los de Daugman.

Para su correcta aplicación se ha de buscar un valor de *thresholding* común para la pupila y el iris, y posteriormente detectar los bordes, por lo que serán necesarios como mínimo 2 iteraciones, la primera para detectar el borde límbico y la segunda para detectar el borde pupilar.

Wildes sugiere empezar por el borde exterior(el límbico), por lo que primero se calculará el umbral apropiado para binarizar la imagen y una vez calculado se usará para detectar los bordes con Canny. Se usó el método *.Canny()* de la librería de *OpenCV* para facilitar el proceso.

Detección del borde límbico

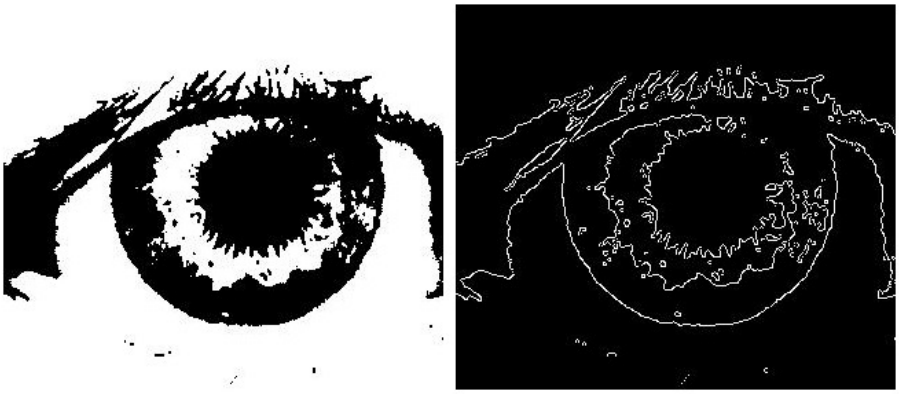

Figura 3.7: Imagen binarizada(*izquierda*). Imagen con bordes detectados con Canny(*derecha*)

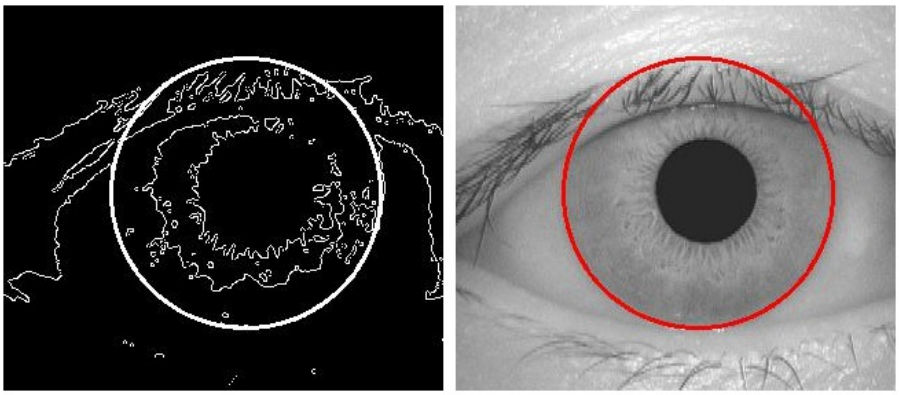

Figura 3.8: Circunferencia exterior dibujada sobre el *edge map* y sobre la muestra original.

El proceso de detección del borde límbico se corresponde con la primera iteración, la detección del borde pupilar, se corresponde con la segunda y los pasos a realizar son iguales, ya que lo único que se necesitará es ajustar los umbrales.

Detección del borde pupilar

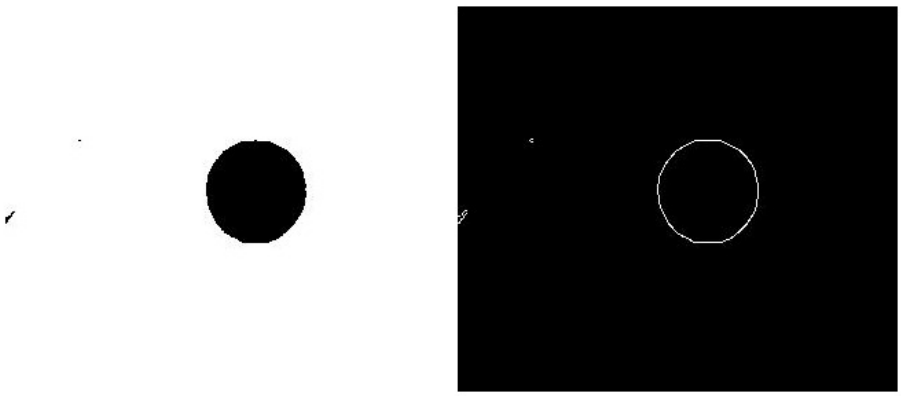

Figura 3.9: Imagen binarizada(*izquierda*). Imagen con bordes detectados con Canny(*derecha*)

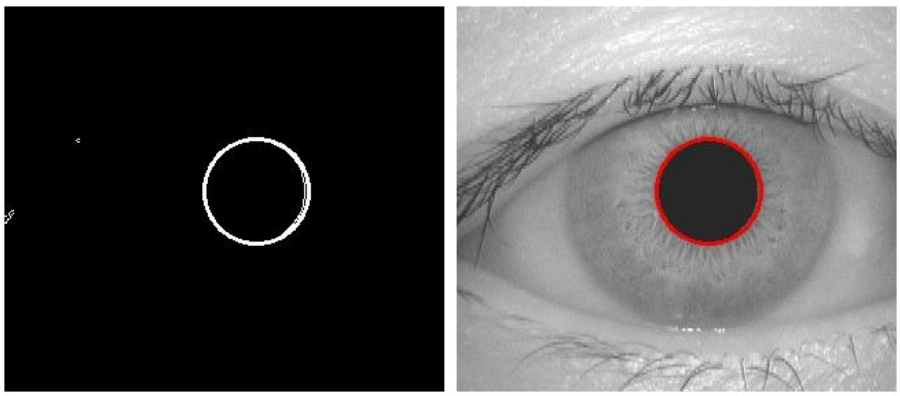

Figura 3.10: Circunferencia interior dibujada sobre el *edge map* y sobre la muestra original.

Para ejemplificar el proceso que se ha de seguir para encontrar los bordes, se ha trabajado sobre la muestra *001_1_1.bmp* de CASIA-Iris-V1, aunque a priori parecía que el proceso desarrollado podría aplicarse a todas las muestras del dataset, más tarde se comprobó que no era así ya que existían fallos principlamente para detectar el borde exterior.

Detección errónea del borde límbico

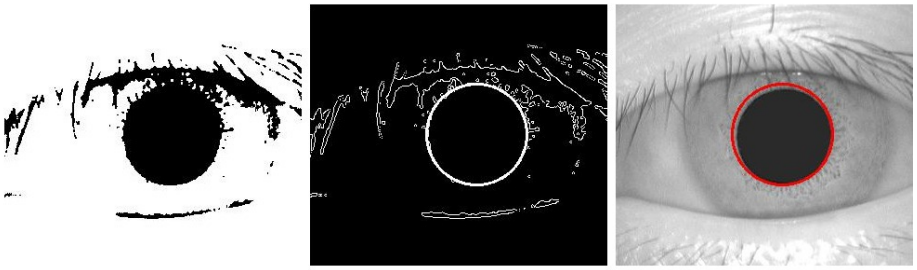

Figura 3.11: Imagen *002_1_1.bmp*, debido a la mala binarización interpreta la pupila como el iris.

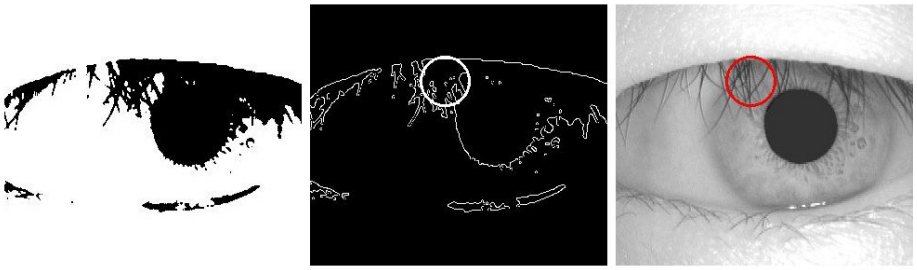

Figura 3.12: Imagen *008_1_1.bmp*, debido a la mala binarización dibuja la circunferencia en una región aleatoria.

Ahora si se quiere detectar el borde pupilar de las 2 muestras anteriores se observa que lo hace correctamente.

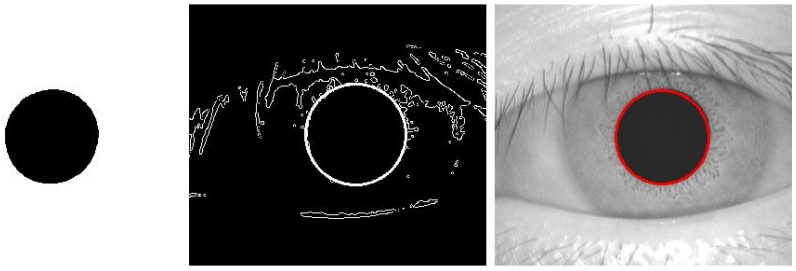

Figura 3.13: Imagen *002_1_1.bmp*, pupila detectada correctamente

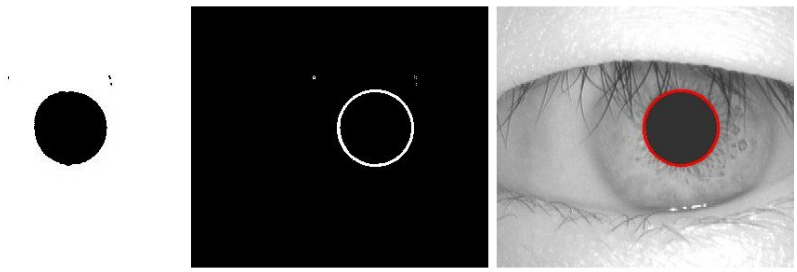

Figura 3.14: Imagen *008_1_1.bmp*, pupila detectada correctamente.

Por lo tanto se llega a la conclusión que es posible aplicar un *thresholding* común para todas las muestras pero únicamente para la detección de la pupila, mientras que para la detección del iris, se deberá ir probando constantemente hasta encontrar el apropiado para cada imagen, esto se correspondería con una técnica de fuerza bruta muy ineficiente, que podría ser factible si se cuenta con un dataset pequeño, pero para el de CASIA que cuenta con 108 muestras no lo es.

Para solucionar este problema se decidió usar el modelo preentrenado *U-Net* que se encargará de la segmentación automática del iris [24, 18].

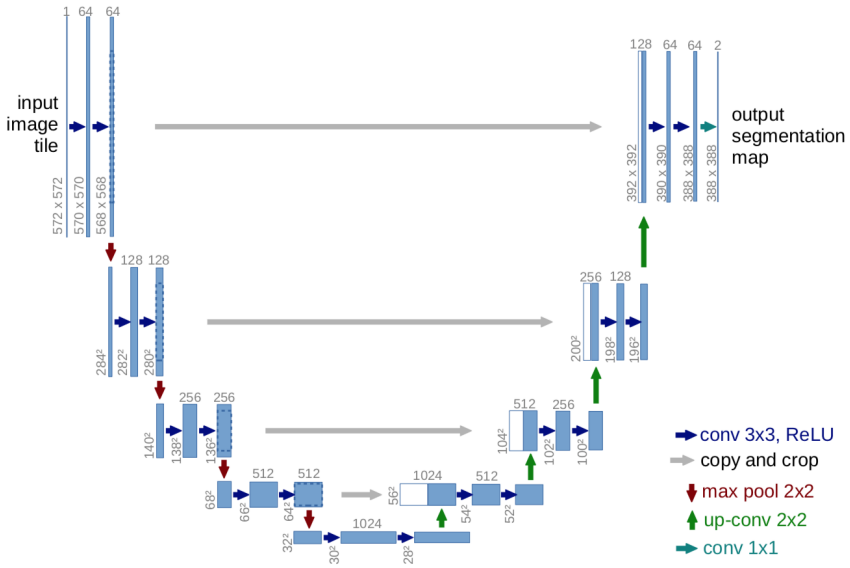

Figura 3.15: El modelo *U-Net* se trata de una Red Neuronal Convolucional usado en problemas biomédicos para la segmentación de diferentes tipos de células y la detección de sus bordes.

La principal ventaja que aporta el uso del modelo mencionado es la capacidad de entrenar modelos precisos a partir de un *dataset* pequeño, lo cuál es un problema muy frecuente en problemas de visión computarizada y en la que por supuesto se incluye la segmentación del iris [6].

El hecho de usar el modelo previamente entrenado aceleró mucho el proceso ya que para usarlo únicamente se necesitaba ajustar la muestra al `input_shape` del modelo y usar el método `predict()` [17]

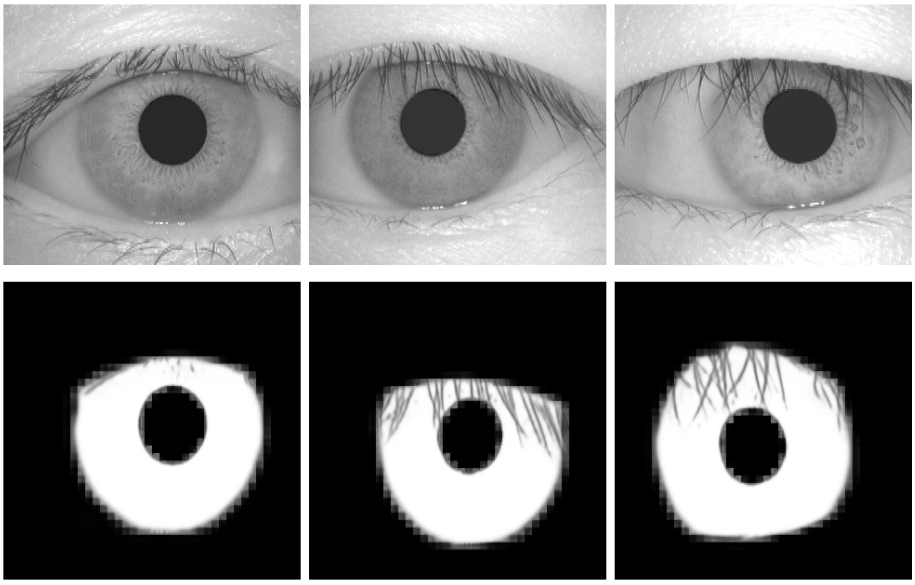

Figura 3.16: Output generado para las muestras *001_1_1.bmp*, *005_2_4.bmp*, *008_2_4.bmp*, con dimensiones 320x320.

El modelo preentrenado segmenta bien los iris, pero como las dimensiones no se correspondían con las de las muestras orginales se las volvió a redimensionar a 320x280 para ya posteriormente aplicar el detector de bordes de Canny.

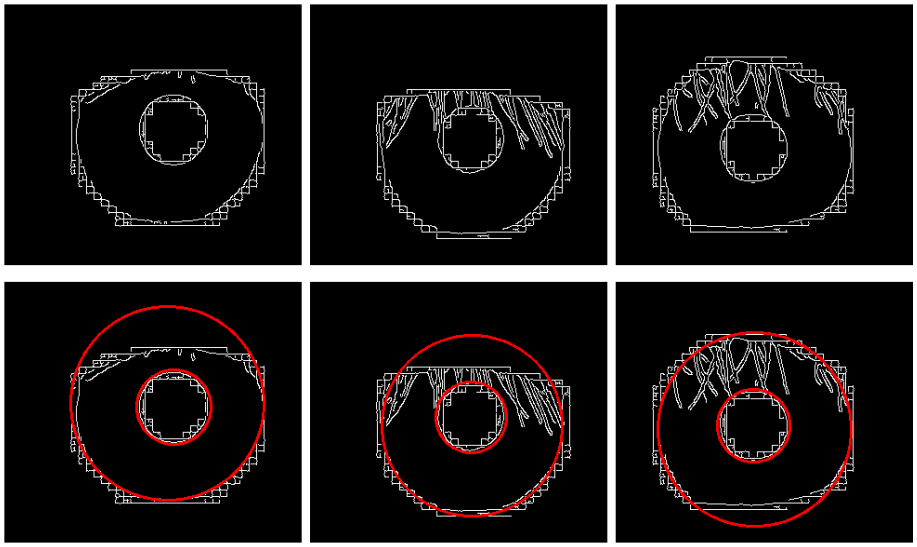

Figura 3.17: Bordes detectados o *edge map* con Canny (arriba). Coordenadas de los bordes límbico y pupilar dibujadas sobre el *edge map* (abajo)

Ahora solo queda dibujar las mismas coordenadas encontradas en las muestras para hacernos una idea de la región con la que trabajaremos en la posterior etapa de normalización.

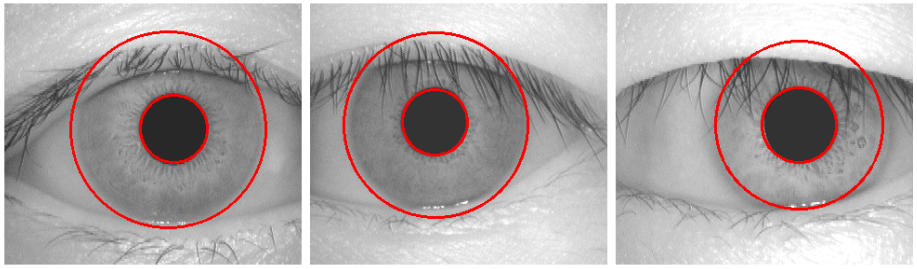

Figura 3.18: Detectados bordes límbico y pupilar.

Llegados a este punto se decidió cambiar los nombres a las muestras para que el etiquetado sea más intuitivo, de modo que se le asignaron nombres reales de un fichero .csv disponible en *Kaggle*, por ejemplo, la muestra *001_1_1.bmp* se renombró con *Adrianna_1*, *005_1_1.bmp* con *Aleah_1* y *005_1_1.bmp* con *Amayah_1*.

Image	pupil cx	pupil cy	pupil radius	iris cx	iris cy	iris radius	Clase
Adrianna_1.bmp	182	134	33	176	135	105	Adrianna
Adrianna_2.bmp	173	138	35	169	140	104	Adrianna
Adrianna_3.bmp	174	120	35	168	121	107	Adrianna
Adrianna_4.bmp	183	122	37	179	122	104	Adrianna
Adrianna_5.bmp	177	145	35	173	149	109	Adrianna
Adrianna_6.bmp	179	133	36	177	136	108	Adrianna
Adrianna_7.bmp	154	131	33	157	137	100	Adrianna
Aharon_1.bmp	181	141	47	158	138	93	Aharon
Aharon_2.bmp	184	139	45	186	134	118	Aharon
Aharon_3.bmp	172	155	44	145	159	90	Aharon

Tabla 3.1: Centros y radios aproximados para cada una de las muestras del dataset CASIA v1.

Normalización

Una vez localizadas las coordenadas del iris en la etapa anterior se lleva a cabo la normalización del iris en la que se genera imágenes del iris, todas ellas con las mismas dimensiones de modo que permita su comparación con otros iris [10, 22, 14]

Este método elimina las incosistencias de tamaño entre las mismas muestras que se pueden generar debido al estrechamiento o ensanchamiento de la pupila ("tendremos menos o más región útil de la que extraer atributos") como respuesta a factores externos como la iluminación.

Para normalizar el iris se usará el método de Daugman, el cual permitirá transformar el iris de coordenadas cartesiana a polares, es decir, transforma la region circular del iris en una rectangular de dimensiones constantes [19]

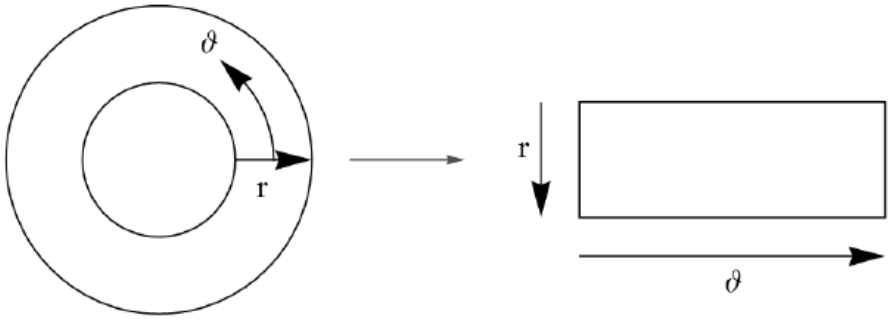

Figura 3.19: Método de Daugman para normalizar. Iris original (izquierda). Iris normalizado (derecha)

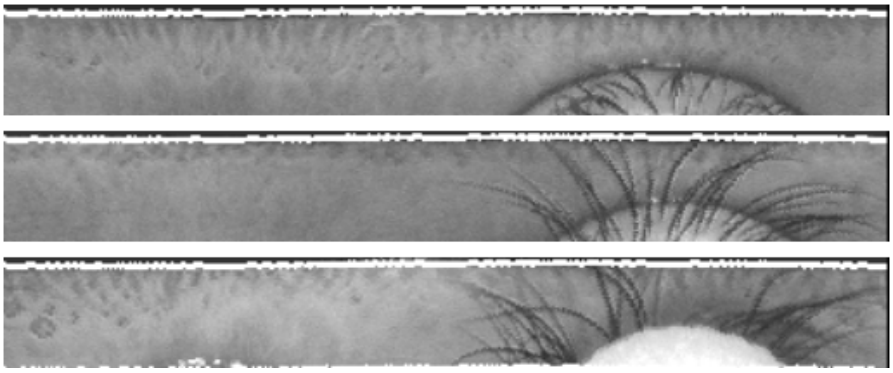

Figura 3.20: Muestras normalizadas, de arriba a abajo, *Adrianna*, *Aleah*, *Amayah*

Se observa que en las muestras normalizadas nos aparecen regiones que no aportan nada, en este caso correspondientes a párpados y pestañas y sólo pueden entorpecer la posterior clasificación, por lo que se ha decidido que la *ROI* será la mitad de la muestra en la que no hay estorbos, ya que con esa región es suficiente para extraer los atributos de la superficie del iris.

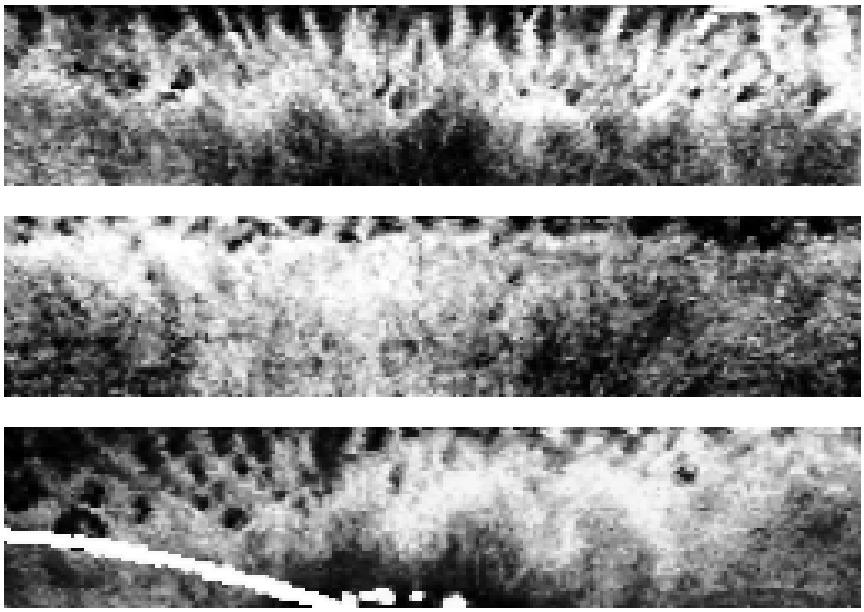

Figura 3.21: *ROI*

Extracción de atributos

Con las muestras ya normalizadas ahora se procede a la extracción de atributos de la superficie del iris, aunque los métodos tradicionales como *Filtros de Gabor* [12] o la *Transformada de Wavelet* [5] entre otros, han sido y son los más ampliamente usados, se ha optado para este proyecto el uso de modelos de *Deep Learning* [20, 6, 14]

Los modelos se tomarán de la biblioteca de *deep learning Keras* [2], la cual ofrece una amplia variedad de modelos implementados y entrenados.

Name	Class	VGG16_0	InceptionV3_0	ResNet50_0
Adrianna_1.bmp	Adrianna	0.183428	0.000724	0.007881
Adrianna_2.bmp	Adrianna	0.148888	0.000720	0.008597
Adrianna_3.bmp	Adrianna	0.138908	0.000677	0.007178
Adrianna_4.bmp	Adrianna	0.123969	0.000734	0.008140
Adrianna_5.bmp	Adrianna	0.157805	0.000744	0.009871
Adrianna_6.bmp	Adrianna	0.154508	0.000719	0.007440
Adrianna_7.bmp	Adrianna	0.164218	0.000720	0.005831
Aharon_1.bmp	Aharon	0.166655	0.000796	0.006227
Aharon_2.bmp	Aharon	0.127383	0.000712	0.007649
Aharon_3.bmp	Aharon	0.201308	0.000809	0.008451

Tabla 3.2: Algunos de los atributos extraídos por cada uno de los modelos de *deep learning* para cada una de las muestras del dataset CASIA v1.

Los atributos extraídos se corresponden con las columnas *VGG16_0*, *InceptionV3_0*, *ResNet50_0*, cabe recalcar que estos no son todos los atributos extraídos ya que dependiendo del modelo elegido para la extracción tendremos tantas columnas como *outputs* tenga el modelo en su útima capa *Fully Connected*.

3.7. Clasificación

Se ha decantado por el uso de estos algoritmos, ya que son más fáciles de comprender que las matemáticas detrás de los métodos tradicionales.

Se usarán los algoritmos de clasificación de la biblioteca de *Scikit Learn* [3].

	SVM	Nearest Neighbors	LogisticRegression	Random Forest
VGG16	0.890212	0.738095	0.904762	0.744709
InceptionV3	0.887566	0.734127	0.912698	0.805556
ResNet50	0.853175	0.710317	0.892857	0.617725

Tabla 3.3: Tasas de acierto

En la tabla anterior se observa que el modelo y el algoritmo que arrojan la mayor precisión, es el *InceptionV3* y el *Logistic Regression*.

Por lo que se usará esa tupla para el *matching*, extraeremos los atributos con *InceptionV3* y clasificaremos el sujeto con el algoritmo de *Logistic Regression* al que entrenaremos con los features extraídos por el modelo de *deep learning*.

El entrenamiento del clasificador se realiza satisfactoriamente y alcanza una precisión siempre por encima del 90 %, por lo que cuando se le pide clasificar (identificar) un sujeto del dataset, lo hace perfectamente.

Sujetos	Probabilidad (%)
Aidan	0.000093
Lindsay	0.000126
Mushka	0.000184
Rafael	0.000140
Scarlett	0.998954

Tabla 3.4: 5 individuos más probables

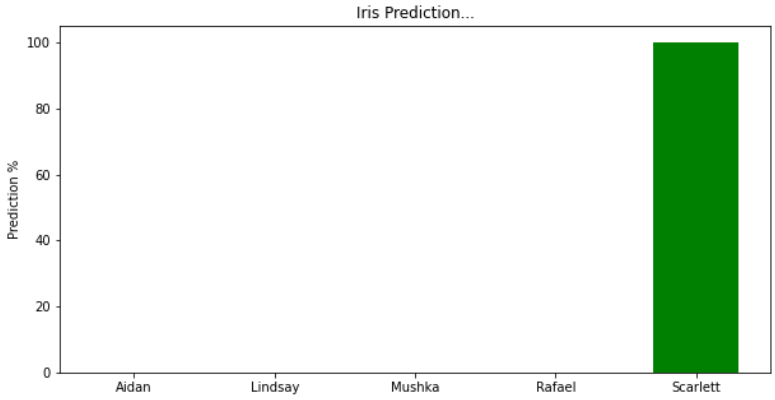

Figura 3.22: Clasificación top 5

Técnicas y herramientas

4.1. Gestión del proyecto

Scrum

Se trata de una metodología ágil para el desarrollo *software*. Se basa en el desarrollo de pequeñas tareas que se revisaran e irán encrementando en un plazo establecido (*sprints)* [31].

Github

Es una plataforma para el alojamiento de proyectos usando el sistema de control de versiones de *git* .

Git

Es un sistema de control de versiones que se encarga de llevar un registro de los cambios en los proyectos y que permite coordinar el trabajo cuando el proyecto es compartido [33].

4.2. Herramientas

Python

Es un lenguaje de programación interpretado, dinámico y orientado a objetos. Se trata de uno de los lenguajes de programación más populares actualmente, entre otras cosas por su corta curva de aprendizaje y su versatilidad lo hace un lenguaje idóneo para el *data science* [36].

Anaconda

Es una suite multiplataforma creado específicamente para el *data science* que incorpora *Jupyter Notebooks* los cuales permiten crear y compartir documentos que contiene código, ecuaciones y facilita la experimentación con *Machine Learning*.

JupyterLab

Se considera la siguiente generación de interfaces basadas en la web para el projecto *Jupyter*. JupyterLab permite trabajar con Jupyter notebooks, editores de texto y terminales. Se eligió esta herramienta como alternativa a los noteboks tradicionales por ser más cómoda.

4.3. Bibliotecas de Python

Numpy

Biblioteca que soporta el trabajo con vectores y matrices de gran tamaño que incluye una gran colección de funciones matemáticas listas par su uso.

Pandas

Se trata de una herramienta *open source* destinada al análisis y manipulación de datos rápida, potente, flexible y fácil de usar.

OpenCV

Es la biblioteca *open source* multipataforma para la visión por computadora por excelencia. Está publicada bajo la licencia BSD 3-Clause que permite que sea usada libremente para propósitos de investigación y comerciales. El hecho de disponga de un buena documentación que se actualiza constantemente es uno de los factores que determina su popularidad [35].

Scikit Image

Biblioteca con una serie de algoritmos para el procesamiento de imágenes.

Scikit Learn

Biblioteca de *Machine Learning* para *Python*. Incluye algoritmos de clasificación, regresión y *clustering* [37].

Keras

Biblioteca de *deep learning* diseñada para la experimentación y el desarrollo con modelos de aprendizaje profundo compatible con los *frameworks* de *TensorFlow*, *MCTK* o *Theano* [34].

TkInter

Binding de la biblioteca gráfica de *Tcl/Tk* considerda el estándar de interfaz gráfica para Python.

4.4. Documentación

LaTeX

Es un sistema de composición de textos orientado a la creación de documentos que presenten una alta calidaD tipográfica [30]. Su amplia gama de posibidades lo hace una herrmienta muy usada en la redacción de artículos y libros científicos que incluyan expreiones matemáticas entre otras cosas.

Texmaker

Es un editor para la redacción de documentos con LaTeX. Para su funcionamiento es necesario la previa instalación de *Tex*. Al final se cambió por *Overleaf* por problemas continuos a la hora de compilar los ficheros *.tex*.

Overleaf

Es un editor de texto online para la redacción de documentos con LaTeX. El hecho de que sea online hace que no sea necesaria ninguna instalación y permite entre otras cosas la colaboración en compartida en tiempo real , control de versiones y trabajar desde cualquier lugar.

Aspectos relevantes del desarrollo del proyecto

5.1. ¿Por qué CASIA?

Si buscamos una base de datos de iris, encontraremos 2 que son las más populares, estas son, CASIA Iris Database [1] y UBIRIS Database [4].

Ambas bases de datos ofrecen imágenes aisladas del iris con las que se podía experinmentar pero podría decirse que las muestras de UBIRIS eran más complejas de tratar, ya que incluía imágenes desenfocadas, con mucho ruido, y algunas en las que incluso se perdía la región que nos interasaba debido a por ejemplo, muestras con ojos cerrados.

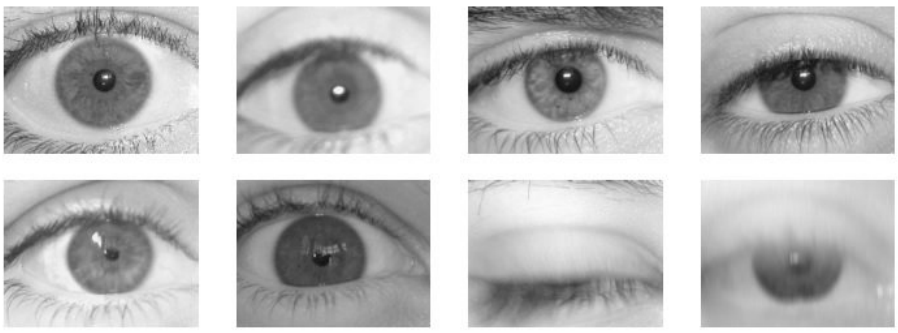

Figura 5.23: Muestras del dataset UBIRIS.v1

En cambio las muestras de CASIA eran uniformes y ya venían preparadas para poder empezar a trabajar con ellas, esto hacía que la etapa de preprocesamiento 3.6 fuese menos tediosa.

Para la descarga del dataset de CASIA, era necesario un registro previo en la página web del CSBR(*Center for Biometrics and Security Research*) así cómo una confirmación por parte de éstos. En cambio para la descarga del de UBIRIS, no se necesitaba nada, sólo accedías y descargabas.

Necesité de 2 intentos para poder registrarme, ya que la primera vez lo intenté con el correo de la universidad pensando que con ello tendría más posibilidades pero nunca me confirmaron, en el segundo intento probé con un correo ordinario de *Gmail* y tuve éxito.

Así que podría decirse que el motivo principal fue la comodidad y también el hecho de que la mayoría de los *papers* en los que me he basado usaban esta base de datos.

5.2. Segmentación manual vs automática

En un principio, para la etapa de segmentación 3.6, no se planeó usar un modelo de *deep learning* pero cOmo se explicó en dicho apartado, se tuvo muchas dificultades para encontrar parámetros que permitieran la localización de los bordes límbico y pupilar para todas las muestras del dataset.

El método tradicional consistía en encontrar un umbral (*thresholding*) global que permitiera binarizar la muestra lo suficientemente bien como para que el detector de bordes de Canny hiciera su trabajo, localizando las 2 circunferencias que nos interesaban, pero como no fue posible, el proyecto tomó la dirección de usar *deep learning* desde etapas tempranas.

Tras la descarga del modelo del repositorio, el proceso para poder usarla fueron los siguientes:

- Se cargaba el modelo con `load_model` de *Keras*

```
from keras.models import load_model

model = load_model('Iris_unet_d5.h5')
model.summary()
```

Figura 5.24: Cargar modelo preentrenado.

- Las muestras originales de CASIA eran de dimensión 320x280, por lo que para pasárselas al método `predict()` del modelo era necesario redimensionar a un *shape* cuadrado, es decir, misma altura y anchura.

No se le pasaron todas las muestras a la vez sino que se implentó una función generadora `test_generator()`.

```
def testGenerator(images, target_size= (320,320)):
    for muestra in images:
        img = muestra / 255
        img = trans.resize(img,target_size)
        img = np.reshape(img,(1,)+img.shape)
        yield img
```

Figura 5.25: Función generadora.

Una vez se tenía el *output* de la red, se volvía a redimensionar a 320x280, pero aunque la segmentación era correcta, las muestras aparecían pixeladas, de modo que se temía que el detector de Canny tuviese problemas para tratar las muestras en ese estado.

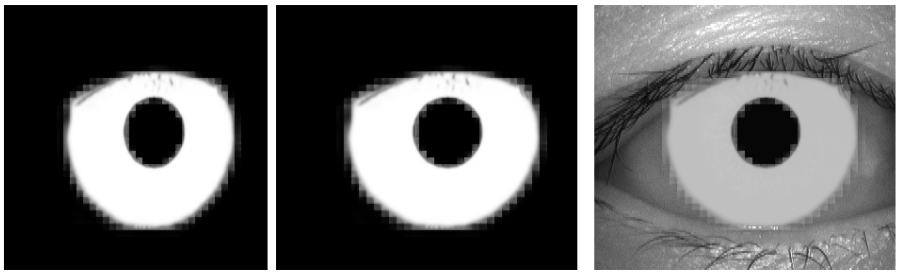

Figura 5.26: Output original (izquierda), Output redimensionado (centro). ROI superpuesto sobre muestra original (derecha).

Pero tal y cómo se observa en la figura 3.17, vemos que ese problema era infundado, de todos modos se quiso eliminar riesgos y se aplicó un filtro *Gaussiano* con el objetivo de suavizar los bordes de las muestras, y después se binarizó nuevamente.

Aunque en 3.6 no se especifica, fue este el método que se siguió como paso previo a la localización de las coordenadas del iris y se aplicó a todas las muestras:

- `cv2.GaussianBlur(img,(17,17),0)`, este método difuminaba la muestra, la clave a la hora de usar dicho método era fijar un *kernel* apropiado (positivo e impar), es decir, cuanto mayor era el valor que se le asignaba, más difuminaba la muestra.

- `cv2.threshold(blur, 70, 255, cv2.THRESH_BINARY)`, en este método era primordial fijar un umbral apropiado, se optó por 70, básicamente lo que hace el método, es que a cualquier pixel con intensidad <70, lo fija a blanco y al resto, >70, a negro, es así como se lleva a cabo el proceso de binarización.

Figura 5.27: *Output* de la red redimensionado (izquierda). Filtro Gaussiano (centro). Segmentación final (derecha).

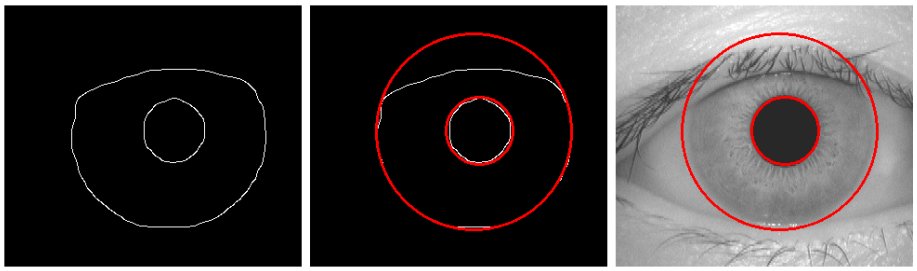

Figura 5.28: Coordenadas localizadas sobre los bordes detectados o *edge map*.

Se optó por este proceso previo en lugar de aplicar Canny directamente sobre los *outputs* redimensionados, porque la región sobre la que se debe aplicar Canny queda más clara ya que nos libramos de estorbos como pueden ser las pestañas.

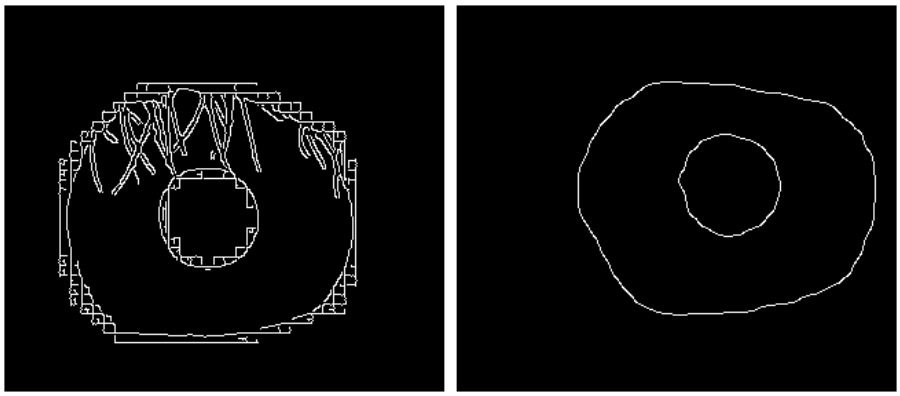

Figura 5.29: Detección de bordes sin filtro Gaussiano (izquierda) y con filtro (derecha) sobre la muestra *008_1_1.bmp*

5.3. *Deep Learning* para la extracción de atributos

Un modelo preentrenado [9] no es más que una red neuronal que previamente ha sido entrenada con un dataset enorme y general, esto hace que los patrones aprendidos por la red pueden hacer que actúe como un modelo genérico capaz de resolver cualquier problema. Esta portabilidad es lo que hace tan poderoso al *deep learning*.

Una *CNN* se compone de 2 partes: la base convolucional y el clasificador. El proceso consistirá en usar la base convolucional para la extracción de los atributos ya que es la parte de la red más general y por tanto reusable, y entrenar un nuevo clasificador con dichos atributos.

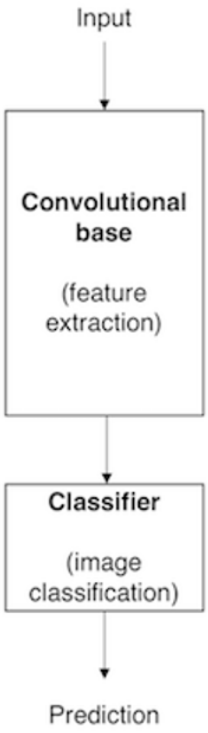

Figura 5.30: Arquitectura de un modelo basado en una Red Neuronal Convolucional [21]

Usaremos el modelo VGG16 [26] el cual se importa desde *Keras*, y ha sido entrenado sobre el dataset de *Imagenet*.

```
# modelo
conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(224, 224, 3))
```

Figura 5.31: Intanciación del modelo VGG16

Los parámetros son:

- `weights`: para que cargue los patrones aprendidos al entrenarse con *Imagenet*.
- `include_top`: booleano que indica si incluímos las últimas capas *Fully Connected*, en nuestro caso no las incluímos ya que esas capas se

corresponderían con las 1000 clases de *Imagenet* pero para nuestro proyecto se necesitarán únicamente 108.

- `input_shape`: dimensiones de los tensores que se le pasarán a la red.

Con la ayuda del módulo `splitfolders` creamos las estructuras de los directorios de entrenamiento y testeo.

```
8  def extract_features(directory, sample_count):
9      features = np.zeros(shape=(sample_count, 7, 7, 512))  # Debe ser igual al output de la conv_base
10     labels = np.zeros(shape=(sample_count,108)) # 108 classes
11     # Preprocess data
12
13     generator = datagen.flow_from_directory(directory,
14                                             target_size=(img_width,img_height),
15                                             batch_size = batch_size,
16                                             class_mode='categorical')
17
18     i = 0
19     for inputs_batch, labels_batch in generator:
20         features_batch = conv_base.predict(inputs_batch)
21         features[i * batch_size: (i + 1) * batch_size] = features_batch
22         labels[i * batch_size: (i + 1) * batch_size] = labels_batch
23         i += 1
24         if i * batch_size >= sample_count:
25             break
26     return features, labels
27
28 train_features, train_labels = extract_features(train_dir, 324)
29 validation_features, validation_labels = extract_features(val_dir, 216)
30 test_features, test_labels = extract_features(test_dir, 216)
```

Figura 5.32: Método para extraer los atributos con la base convulocional.

En la línea 9 se especifica el *output shape* de la última capa de la base convolucional, y en la 10, las muestras que nos interesan, que serán 108 individuos. En la línea 11 creamos un generador y con `.flow_from_directory()` las generamos desde los directorios creados anteriormente, importante especificar `class_mode='categorical'` ya que se trata de un problema de clasificación multiclase. Por último en la línea 20 usamos el método `.predict()` de la base convolucional para extraer los atributos.

Ahora se creará el nuevo clasificador, se compilará y se entrenará.

```
1 model = Sequential()
2 model.add(GlobalAveragePooling2D(input_shape=(7,7,512)))
3 model.add(Dense(108, activation='softmax'))
4 model.compile(loss='categorical_crossentropy',
5               optimizer='adam',
6               metrics=['accuracy'])
```

Figura 5.33: Creación y compilación del nuevo clasificador adecuado para los 108 individuos.

```
history = model.fit(train_features, train_labels,
                    epochs=150,
                    batch_size=32,
                    validation_data=(validation_features, validation_labels))
```

Figura 5.34: Entrenamiento del clasificador.

```
Epoch 145/150
11/11 [==============================] - 0s 8ms/step - loss: 2.4015 - accuracy: 0.9383 - val_loss: 2.9801 - val_accuracy: 0.6435
Epoch 146/150
11/11 [==============================] - 0s 9ms/step - loss: 2.3909 - accuracy: 0.9290 - val_loss: 2.9730 - val_accuracy: 0.6481
Epoch 147/150
11/11 [==============================] - 0s 11ms/step - loss: 2.3808 - accuracy: 0.9228 - val_loss: 2.9653 - val_accuracy: 0.6481
Epoch 148/150
11/11 [==============================] - 0s 9ms/step - loss: 2.3713 - accuracy: 0.9167 - val_loss: 2.9580 - val_accuracy: 0.6528
Epoch 149/150
11/11 [==============================] - 0s 9ms/step - loss: 2.3607 - accuracy: 0.9321 - val_loss: 2.9501 - val_accuracy: 0.6528
Epoch 150/150
11/11 [==============================] - 0s 8ms/step - loss: 2.3502 - accuracy: 0.9290 - val_loss: 2.9428 - val_accuracy: 0.6574
```

Figura 5.35: Últimas 5 *epochs* del entrenamiento.

Tal y cómo se observa se alcanza un acierto del 65 % por lo que se decide aumentar dicho porcentaje cambiando nuevamente de enfoque.

Ahora se usarán 3 modelos para la extracción de atributos, que serán: *VGG16*, *InceptionV3* y *ResNet50* y se usarán clasificadores de *Sckikit Learn*.

Todo este proceso viene redactado en el notebook `#8 Feature extaction-v2.ipynb`

```
models_dict = {}
vgg16_dict, inception_v3_dict, resnet50_dict = {} ,{} ,{}


print("Loading VGG16")
model = VGG16(weights='imagenet')
model = Model(model.input, model.layers[-2].output) # output_shape de la penúltima capa(fully-connected) -> (,4096)
vgg16_dict["model"] = clone_model(model)
vgg16_dict["preprocesor"] = vgg16_preprocessor
vgg16_dict["target_size"] = model.input_shape[1], model.input_shape[2]# default vgg16 input (224,224)
```

Figura 5.36: Ejemplo de carga de uno de los 3 modelos, para los 2 restantes el proceso es igual, se cargan y se almacenan en un diccionario de modelos (`models_dict()`).

Para la extracción de atributos se usará el fichero *iris_data.csv* generado en el notebook `#6 Funciones-v2.ipynb` en el que se guardan las coordenadas de los bordes límbico y pupilar y las muestras ya polarizadas que estarán guardadas en el directorio *CASIA-Polar*. Ver 3.1.

Destacan 3 funciones:

- `extract_features(image_path,model_name)`: es la función encargada de extraer los atributos de la muestra polarizada, se le pasa la ruta de la muestra y el modelo que se desea usar para la extracción.

- `register_to_deepfeatures(register)`. guarda los atributos extraídos de las muestras polarizadas en filas de un *DataFrame*.
- `DataFrame.apply()`: aplica la función `extract_features()` a todas las filas del fichero *iris_data.csv*, parecido a `map()`.

Una vez realizado lo anterior tendremos como resultado el fichero *iris_features.csv* 3.2 y se pasará a elegir los clasificadores de Scikit Learn. Se eligieron 4:

- Support Vector Machine (SVM)
- Logistic Regression
- Nearest Neighbours
- Random Forest

Como en un mismo fichero tenemos todos los atributos extraídos por los 3 modelos, se procederá a crear 3 ficheros independientes que guardasen los features de cada modelo. Con esto se consigue X e y.

Se creará una función `cross_validate_preds_model(X, y, model, num_folds)`: para evaluar los clasificadores con evaluación cruzada. Lo único destacable de esta parte es el *num_folds* que se le ha fijado a 4 por ser divisor de 756. Definir ese número de *folds* permitirá al modelo entrenarse en 4 versiones (de 189 muestras cada una) del set de entrenamiento y evaluarse en 4 versiones del set de testeo.

El resultado es el visto en la tabla 3.3.

Por lo que se llega a la conclusión de que el mejor modelo de *deep learning* y el mejor clasificador para esta tarea en específico son: *InceptionV3* y *Logistic Regression*.

A la hora de usar el método `predict()` del clasificador observamos que efectivamente funciona e identifica a los sujetos perfectamente. Su rendimiento es muy bueno, ya que incluso cuando hacemos una clasificación *top-5* con `predict_proba()`, vemos la probabilidad con la que identifica al sujeto correcto respecto a los otros 4 más probables 3.22.

El clasificador se serializa (*logistic_clf_trained.pkl*) y se guarda para usarlo en la aplicación de escritorio.

Trabajos relacionados

6.1. Artículos

Diagnóstico de la diabetes mediante el análisis del iris

Se trata de un proyecto de investigación realizado por los alumnos del bachillerato de excelencia Raúl Santamaría, Jana Bisabarros y Estrella Higuera en el año 2017.

Su estudio se centra en averiguar si los patrones del iris pueden delatar indicios de diabetes en pacientes. Aunque el objetivo final es distinto al estudiado en este proyecto, puede observarse ciertas etapas comunes a ambos proyectos, como pueden ser la fase de segmentación y normalización. Las metodologías para realizar dichas etapas son distintas pero sirvieron de inspiración al principio del desarrollo de este proyecto cuando se tenía dudas de como empezar.

Image Understanding for Iris Biometrics: A Survey

Es un estudio por parte de Kevin Bowyer, Karen Hollingsworth y Patrick Flynn, en la que se profundiza en el estado del arte de los métodos existentes para el uso del iris como biometría. Destacar que se refiere a los métodos tradicionales, es decir, a los matemáticos, y no a los nuevos que incluyen técnicas de *Machine Learning*.

A multi-biometric iris recognition system based on a deep learning approach

Alaa S. Al-Waisy, Rami Qahwaji, Stanley Ipson, Shumoos Al-Fahsawi y Tarek A.M. Nagen son los autores de este *paper* en el que se propone el uso de modelos de *deep learning* para el reconocimiento del iris, cabe destacar que no es el primero en el que se propone este nuevo enfoque.

Desarrollan un modelo de *deep learning* propio llamado *IrisConvNet* para el entrenamiento con 3 bases de datos de iris distintas obteniendo un sistema de reconocimiento con una tasa de acierto del 100 % en menos de un segundo.

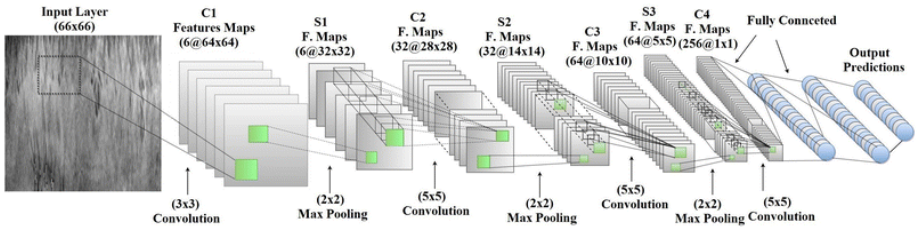

Figura 6.37: Arquitectura de *IrisConvNet*

Conclusiones y Líneas de trabajo futuras

Conclusión

Tras finalizar el proyecto se llega a la siguiente serie de conclusiones:

- El objetivo general que consistía en estudiar las fases del reconocimiento e implementar un modelo funcional propio se ha cumplido. Por supuesto hubo complicaciones que hicieron que el proyecto se desarrollase más lentamente y estuviese incluso a punto de abandonarse, pero que pudieron superarse gracias a la extensa documentación disponible en internet y a la propia ayuda del tutor.

- El proyecto ha requerido de los conocimientos adquiridos en la carrera y de unos nuevos, por lo que el hecho de haber realizado este proyecto ha servido para formarse y adquirir competencias de manera autodidacta.

- El hecho de haber elegido temáticas de *Machine Learning* y *Deep Learning* ha permitido percatarse de la verdadera magnitud de dichas tecnologías en el mundo actual y nos da pistas de la dirección que tomarán todo tipo de industrias en el futuro.

- Por muy bien que esté planificado un proyecto, siempre se ha de estar preparado para los inconvenientes que surgan y se debe ser capaz de redirigirlo de modo que no se ponga en peligro su correcto desarrollo.

Líneas de trabajo futuras

- Aunque en el apartado de conclusiones se dice que el proyecto se ha desarrollado correctamente, esto no es así. En la última etapa a la hora de exportar el clasificador entrenado para usarlo en la aplicación de escritorio, hubo problemas que no dio tiempo a solucionar.

  Al entrenar el clasificador en los *notebooks* con los datos del fichero `iris_features.csv` se obtiene un modelo funcional que al ponerse a prueba, funciona a la perfección, es decir, clasifica correctamente. Pero cuando se usa en la aplicación empieza a fallar estrepitosamente, ya que clasifica incorrectamente y por lo general el resultado siempre es *Xin* o *Anas*.

  Es por ello que se propone como tarea investigar y solucionar este error, ya que es el único destacable, las etapas anteriores aunque igual no son muy óptimas, cumplen con su labor.

- Convendría ampliar el conjunto de muestras sobre la que es aplicable el proyecto, es decir, debería hacerse más general de modo que permita clasificar muestras de cualquier dataset de ojos o puede que hasta datasets propios, por ejemplo, muestras tomadas por cualquier usuario, aunque se deberá tener en cuenta lo visto en 3.5.

- Otro aspecto a mejorar sería la funcionalidad de la aplicación de escritorio, ya que la actual es muy básica.

Bibliografía

[1] Casia iris v1. [Online; Accedido 27-Junio-2020].

[2] Keras. [Online; Accedido 18-Julio-2020].

[3] Scikit learn. [Online; Accedido 16-Julio-2020].

[4] Ubiris database. [Online; Accedido 27-Junio-2020].

[5] Hadeel Abdullah and Ahmed Abdullah. Iris recognition using wavelet transform and artificial neural networks. *Engineering and Technology Journal*, 33:877–888, 01 2015.

[6] Alaa Al-Waisy, Rami Qahwaji, Stanley Ipson, Shumoos Al-Fahdawi, and Tarek Nagem. A multi-biometric iris recognition system based on a deep learning approach. *Pattern Analysis and Applications*, 10 2017.

[7] Biometrics. Identificación biométrica a través del iris ocular, 2015. [Online; Accedido 26-Mayo-2020].

[8] Kevin Bowyer, Karen Hollingsworth, and Patrick Flynn. Image understanding for iris biometrics: A survey. *Computer Vision and Image Understanding*, 110:281–307, 05 2008.

[9] Francois Chollet. *Deep Learning with Python*. Manning Publications, 2017.

[10] J. G. Daugman. High confidence visual recognition of persons by a test of statistical independence. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(11):1148–1161, November 1993.

[11] Shiren Y. Elhabian. Iris recognition, 2009. [Online; Accedido 29-Mayo-2020].

[12] D. Gabor. Theory of communication. part 1: The analysis of information. *Journal of the Institution of Electrical Engineers - Part III: Radio and Communication Engineering*, 93(26):429–441, 1946.

[13] Sanjay Ganorkar and Ashok Ghatol. Iris recognition: an emerging biometric technology. pages 91–96, 02 2007.

[14] Álvar Arnaiz-González Javier Ruiz-Pérez Débora Zurro José-Francisco Díez-Pastor, Pedro Latorre-Carmona. Automatic phytolith classification. https://github.com/alvarag/AutomaticPhytolithClassification, 2020. [Online; Accedido 16-julio-2020].

[15] Todor Kazakov. *Iris Detection and Normalization*. PhD thesis, 01 2011.

[16] S.V. Kumar, Nishanth R., Nidhin Sani, Abin Joseph, and Agath Martin. Specular reflection removal using morphological filtering for accurate iris recognition. pages 1–4, 03 2019.

[17] Juz Lozeg. U-net iris segmentation. https://github.com/jus390/U-net-Iris-segmentation, 2019. [Online; Accedido 20-julio-2020].

[18] Jus Lozej, Blaž Meden, Vitomir Štruc, and Peter Peer. End-to-end iris segmentation using u-net. pages 1–6, 07 2018.

[19] Patricio Baldino Lucas Terissi, Lucas Cipollone. Sistema de reconocimiento de iris. 2000.

[20] Elrefaei Maram.G Alaslni, Lamiaa A. Transfer learning with convolutional neural networks for iris recognition. 2019.

[21] Pedro Marcelino. Transfer learning from pre-trained models, 2018. [Online; Accedido 29-Julio-2020].

[22] Akshata Patel. Iris-recognition. https://github.com/akshatapatel/Iris-Recognition, 2019. [Online; Accedido 20-julio-2020].

[23] Estrella Higuera Raúl Santamaría, Jana Bisabarros. Diagnóstico de la diabetes mediante el análisis del iris, 2017. [Online; Accedido 3-Junio-2020].

[24] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. volume 9351, pages 234–241, 10 2015.

[25] Sunil Sandhu. How neural networks process input data, 2020. [Online; Accedido 10-Junio-2020].

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv 1409.1556*, 09 2014.

[27] Aldo Valdez Alvarado. IntroducciÓn al machine learning. 05 2018.

[28] Jorge Valverde-Rebaza. Detección de bordes mediante el algoritmo de canny. 10 2007.

[29] Wikipedia. Latex — wikipedia, la enciclopedia libre, 2012. [Online; Accedido 26-Mayo-2020].

[30] Wikipedia. Latex — wikipedia, la enciclopedia libre, 2015. [Internet; descargado 30-septiembre-2015].

[31] Wikipedia. Scrum, 2018. [Online; Accedido 26-Mayo-2020].

[32] Wikipedia. Aprendizaje automático, 2020. [Online; Accedido 10-Junio-2020].

[33] Wikipedia. Git, 2020. [Online; Accedido 26-Mayo-2020].

[34] Wikipedia. Keras, 2020. [Online; Accedido 26-Mayo-2020].

[35] Wikipedia. Opencv, 2020. [Online; Accedido 26-Mayo-2020].

[36] Wikipedia. Python, 2020. [Online; Accedido 26-Mayo-2020].

[37] Wikipedia. Scikit-learn, 2020. [Online; Accedido 26-Mayo-2020].

[38] Sebastián Yonekura. *Evaluación y mejora de un sistema de reconocimiento de iris a distancia utilizando cámara de alta resolución.* PhD thesis, 2014.