UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

**TFG del Grado en Ingeniería Informática**

# Sistema clasificador de iris
# Documentación Técnica

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

# Índice general

# Índice de figuras

# Índice de tablas

*Apéndice* $A$

# Plan de Proyecto Software

## A.1. Introducción

En el siguiente apartado se detallarán las fases seguidas desde el comienzo hasta la finalización del proyecto, así mismo se realizará una estimación aproximada de de los recursos necesarios que permitirían desarrollar este proyecto en un escenario real.

## A.2. Planificación temporal

La idea de este TFG fue presentada a mi tutor en el mes de febrero, pero no se empezó a trabajar en él como tal hasta el mes de junio. Al principio se planearon revisiones semanales en las que mediante videollamada comunicaba a mi tutor el progreso que había hecho durante esa semana, aunque posteriormente se intercalaron con reuniones bisemanales. Con esto se intentaba aplicar la metodología *Scrum*, en el que cada *sprint* era de una semana, aunque la duración de estos es algo que varió a lo largo del desarrollo del proyecto.

Todas las tareas, avances y cambios del proyecto están reflejados en el siguiente repositorio: https://github.com/jaa0124/iris_classifier

El proyecto constaba de 3 fases bien diferenciadas

- **Fase 1**: Tras discutir la temática que se quería tratar, esta fase se dedicó a la investigación de las técnicas existentes y artículos relacionados.

- **Fase 2**: Tras la fase 1, se llegó a la conclusión de que los conocimientos que se tenían sobre el tema eran escasos, por consiguiente esta fase se dedicó a la autoformación mediante apuntes de asignaturas de la universidad facilitados por el tutor o con cursos de plataformas didácticas como *Udemy* o *Coursera*.
- **Fase 3**: Aplicamos los conocimientos adquiridos y se empieza con la experimentación en cuadernos de *Jupyter* y posteriormente se desarrolla una aplicación de escritorio funcional que resuma y ponga a prueba las conclusiones de los cuadernos.

Cabe destacar que tanto la Fase 1 como la 2, nunca se dieron por completadas, ya que constantemente se necesitaban competencias de las que no te dabas cuenta hasta que te encontrabas con un problema que no podías resolver.

## Sprint 1 (12/06/20 - 19/06/20)

Marca el comienzo del proyecto, como primera tarea se me asigna buscar documentación relacionada con el reconocimiento del iris y encontrar algún método que sea aplicable a lo que se deseaba hacer, así mismo se me facilita:

- un proyecto realizado por alumnos años antes cuya temática era similar aunque su objetivo final era distinto.
- una serie de apuntes de la asignatura de *Minería de datos*.
- *notebooks* de experimentación del propio tutor que serían de mucha utilidad pero en fases más avanzadas del proyecto, aún así, se le dedicó tiempo a su comprensión.

## Sprint 2 (19/06/20 - 26/06/20)

No se asignaron nuevas tareas ya que aún no se habían completado las de la semana anterior debido a la complejidad de algunas, por lo que se solicitó una prórroga de una semana para intentar completar dichas tareas.

## Sprint 3 (26/06/20 - 03/07/20)

Se comienza con la experimentación en los *notebooks*, se solicitaron las siguientes tareas:

- Buscar un dataset con el que realizar los experimentos.

- Una vez elegido, probar posibles métodos para la segmentación del iris.
  - Binarizar las muestras aplicando un *thresholding*.
  - Aplicar el detector de bordes de Canny.

## Sprint 4 (03/07/20 - 10/07/20)

Tras intentar segmentar el iris con las técnicas propuestas por el tutor, no se consigue el resultado esperado, por lo que se me asigna otra prórroga de una semana.

## Sprint 5 (10/07/20 - 17/07/20)

Aunque la idea era presentar el TFG en la segunda convocatoria del mes de julio, se desechó la idea ya que el proyecto estaba en etapas muy tempranas de su desarrollo y se le comunicó al tutor la idea de entregarlo en septiembre y posiblemente cambiar la temática del TFG ya que se llegó a un punto en el que no había avances. El proyecto entra en un *hiatus*.

## Sprint 5 (27/07/20 - 10/08/20)

Se decide continuar con el proyecto, así mismo se me propone un cambio radical para conseguir segmentar el iris: usar un modelo de *deep learning*. Tareas de este *sprint*:

- Probar la red preentrenada *U-Net* para segmentar las muestras automáticamente.
- Mejorar los resultados de la red aplicando operadores morfológicos que faciliten la detección de los bordes.
- Empezar con el proceso de normalización.
- Seleccionar clasificadores de *Scikit Learn*.

## Sprint 6 (10/08/20 - 25/08/20)

La etapa de preprocesamiento de las muestras se había completado con éxito, por lo que ahora quedaba la última, la clasificación.

Para ello se me vuelve a recomendar revisar los *notebooks* del tutor mencionados anteriormente A.2 y aplicar un proceso similar al que aparece

en ellos pero haciendo los cambios que pudieran ser necesarios para que funcione con el proyecto.

## Sprint 7 (25/08/20 - 07/09/20)

Los experimentos en los *notebooks* funcionan perfectamente por lo que se habla de desarrollar un aplicación de escritorio que refleje los resultados, así mismo se comienza a redactar la memoria.

Tareas asignadas:

- Investigar sobre bibiotecas de *Python* para la creación de interfaces de usuario, se recomienda *Kivy* y *TkInter*.
- Empezar a redactar la memoria y enviársela al tutor.

## Sprint 8 (07/09/20 - 21/09/20)

Se tiene preparada una versión de la aplicación y se la muestra al tutor, este *sprint* se dedica a ultimar detalles de funcionalidad de la aplicación y a la finalización de la memoria.

## Sprint 8 (21/09/20 - 28/09/20)

Tareas:

- Organizar el repositorio.
- Entregar TFG.

Con el *sprint 8* damos por concluído el proyecto. Una aclaración importante: estos *sprints* no quedan reflejados como tal en el repositorio porque en un principio se empezó con uno que sí estaba organizado de este modo, pero tras el *hiatus* mencionado se eliminó dicho repositorio y se empezó uno nuevo.

# A.3. Estudio de viabilidad

## Viabilidad económica

### Costes

### Coste Software

Los recursos usados en el proyecto son *open source* y están disponibles de manera gratuita al público, por lo que cualquiera podría acceder a estos.

Concluimos que en este apartado no se tiene coste alguno.

### Coste Hardware

El hardware empleado ha sido un ordenador portátil que se compró específicamente para cursar el grado, valorado en 1200€, teniendo en cuenta que un alumno promedio tarda 5 años en finalizar la carrera, definiremos este número como el tiempo de amortización:

$$\frac{1200}{12 * 5} * 5 = 100€ \tag{A.1}$$

### Coste de formación

Se necesitaban unos conocimientos con los que no se contaba para realizar el proyecto , la carrera ofrecía 3 asignaturas que hubiesen sido de utilidad, pero de las que sólo me matriculé en una por ser la obligatoria. Estas asignaturas eran:

- Sistemas inteligentes
- Computación neuronal y evolutiva
- Minería de datos

La asignatura constaba de 6 créditos a 20€ cada uno, por lo que de haber optado por matricularme en esas asignaturas optativas, el gasto hubiese sido de 240€.

Sin embargo debido a que matricularse en más asignaturas supondría una carga y pondrían en riesgo aquellas en las que ya se estaba matriculado, se optó por usar cursos de plataformas didácticas, se usaron 2:

- **Coursera**:

- *Neural Networks and Deep Learning:* https://www.coursera.org/learn/neural-networks-deep-learning/home/welcome
- *Convulotional Neural Networks:* https://www.coursera.org/learn/convolutional-neural-networks/home/welcome

La suscripción en esta plataforma es mensual, de modo que de estar suscritos el gasto sería de 41€ mensuales, pero en este caso se aprovechó la suscripción gratuita que ofrece Coursera a entidades universitarias debido a las pandemia. Así que el gasto sería 0€.

- **Udemy**
  - *Complete Machine Learning and Data Science: Zero to Mastery:* https://www.udemy.com/course/complete-machine-learning-and-data-sci

  Para este curso de desembolsó una cantidad de 11,99€.

Por lo que el gasto para la formación se queda en 11,99€.

### Coste de personal

Se considera que el proyecto ha sido desarrollado por un investigador de la universidad, que perfectamente se podría completar en unos 4 meses, de modo que:

| Concepto | Coste (€) |
|---|---|
| Salario bruto del trabajador | 1300 |
| Contingencias comunes (23,6 %) | 306,8 |
| Desempleo (5,5 %) | 71,5 |
| FOGASA (0,2 %) | 2,6 |
| Coste total mensual | 1680,9 |

Y teniendo en cuenta que tardará 4 meses en finalizar el proyecto, los gastos ascenderían a 6723,6€.

### Ingresos/beneficios

Se trata de un proyecto de investigación que será publicado de manera gratuita, por lo que en ese aspecto no se espera ningún beneficio económico, en cuanto a los ingresos, podría recibirse alguna financiación o beca.

## Viabilidad legal

Tanto los *papers* en los que se ha basado el proyecto como las herramientas usadas (detalladas en A.3) son de dominio público y son fácilmente accesibles.

| Librería | Versión | Descripción | Licencia |
|---|---|---|---|
| TensorFlow | 2.3.1 | Biblioteca para *Machine Learning*. | Apache |
| Keras | 2.4.3 | Biblioteca para *Deep Learning*. | MIT |
| OpenCV | 4.4.0.44 | Biblioteca de visión por computador. | BSD |
| Scikit-Learn | 0.22.1 | Biblioteca para aprendizaje automático en Python. | BSD |
| VsCode | 1.46.1 | Editor de código. | MIT |
| Matplotlib | 3.3.2 | Biblioteca para generar gráficos. | BSD |
| Numpy | 1.18.5 | Biblioteca para cálculo numérico. | BSD |
| Pandas | 1.1.2 | Biblioteca para análisis de datos. | Libre |
| Jupyter Lab | 2.2.8 | Entorno web de programación. | BSD |
| Jupyter Notebook | 6.0.3 | Aplicación web para *data science* | BSD |
| TkInter | 1.1.2 | Biblioteca para GUI. | Libre |

Tabla A.1: Licencias de bibliotecas.

Así mismo, la base de datos usada tiene fines educativos o de investigación, por lo que se podrá acceder a ella mediante previo registro, pero se prohíbe una distribución alterada de dicha base datos y su uso en la que no se referencia al CASIA (*Chinese Academy of Sciences' Institute of Automation*) y a `http://biometrics.idealtest.org/`

*Apéndice* $B$

# Especificación de Requisitos

## B.1. Introducción

Se procederá a definir los requisitos y objetivos que el proyecto deberá cumplir si se quiere que sea considerado como exitoso.

## B.2. Objetivos generales

Se desea crear un sistema de reconocimiento que use el iris como rasgo biométrico así como desarrollar una aplicación de juguete que ejemplifique su uso en un escenario real.

## B.3. Catálogo de requisitos

### Requisitos funcionales

La aplicación deberá:

- **RF-1:** ser capaz de clasificar (identificar) las muestras de ojos.
- **RF-2:** permitir al usuario ver las etapas por las que ha pasado la muestra antes de ser clasificada.
  - **RF-2.1:** permitir ver la muestra segmentada.
  - **RF-2.2:** permitir ver las coordenadas del borde límbico y pupilar.
  - **RF-2.3:** permitir ver la muestra normalizada.

- **RF-3:** Permitir al usuario elegir la muestra.

  - **RF-3.1:** Debe permitir al usuario navegar en su sistema de ficheros.
  - **RF-3.2:** El usuario podrá cargar las muestras que el desee sin necesidad de reiniciar la aplicación.

### Requisitos no funcionales

La aplicación deberá:

- **RNF-1:** realizar la clasificación en un lapso de tiempo pequeño (buenos tiempos de respuesta).
- **RNF-2:** ser intuitiva y fácil de usar.

## B.4. Especificación de requisitos

### Diagrama de casos de uso

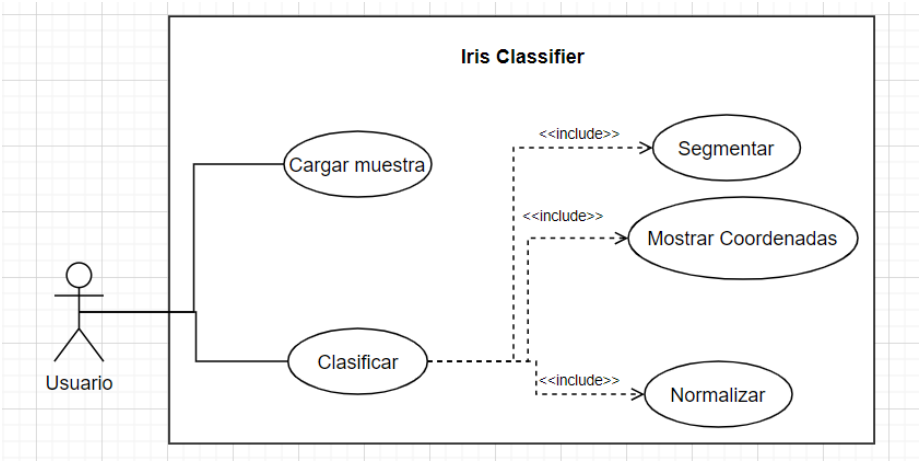


Figura B.1: Diagrama de casos de uso

| CU-1: Cargar muestra | | |
|---|---|---|
| Descripción | Permite al usuario cargar la imagen que quiera de su sistema de archivos. | |
| Pre-condiciones | La imagen corresponde a una muestra del dataset CASIA V1 | |
| | La muestra está guardado en el computador del usuario. | |
| Requisitos | RF-3, RF-3.1, RF-3.2 | |
| Secuencia normal | Paso | Acción |
| | 1 | El usuario abre la aplicación. |
| | 2 | El usuario selecciona «Cargar muestra». |
| | 3 | El usuario elige la muestra que quiera. |
| Postcondiciones | La imagen aparece en la aplicación. | |
| Excepciones | 1 | La muestra no pertence al dataset CASIA V1. |
| Frecuencia | Alta | |
| Importancia | Alta | |

Tabla B.1: Caso de uso 1: Cargar muestra

| CU-2: Clasificar | | |
|---|---|---|
| Descripción | Permite al usuario clasificar la muestra cargada. | |
| Pre-condiciones | Deberá haberse cargado la muestra. | |
| Requisitos | RF-1, RF-2, RF-2.1, RF-2.2, RF-2.3 | |
| Secuencia normal | Paso | Acción |
| | 1 | El usuario selecciona «Clasificar» |
| Post-condiciones | Aparece el sujeto clasificado | |
| | Aparece el botón «Segmentar» | |
| | Aparece el botón «Coordenadas» | |
| | Aparece el botón «Normalizar» | |
| Excepciones | 1 | No se ha cargado ninguna muestra. |
| Frecuencia | Alta | |
| Importancia | Alta | |

Tabla B.2: Caso de uso 2: Clasificar

| CU-3: Segmentar | | |
|---|---|---|
| Descripción | Permite al usuario ver la muestra segmentada | |
| Pre-condiciones | Deberá haberse pulsado «Clasificar». | |
| Requisitos | RF-1, RF-2, RF-2.1 | |
| Secuencia normal | Paso | Acción |
| | 1 | El usuario selecciona «Segmentar». |
| | 2 | Se abre la ventana que muestra la imagen segmentada. |
| | 3 | El usuario cierra la pestaña. |
| Post-condiciones | Aparece la muestra segmentada. | |
| Excepciones | 1 | No se ha clasificado correctamente el sujeto. |
| Frecuencia | Alta | |
| Importancia | Media | |

Tabla B.3: Caso de uso 3: Segmentar

| CU-4: Mostrar coordenadas | | |
|---|---|---|
| Descripción | Permite al usuario ver los borde límbico y pupilar dibujados sobre la muestra. | |
| Pre-condiciones | Deberá haberse pulsado «Clasificar». | |
| Requisitos | RF-1, RF-2, RF-2.2 | |
| Secuencia normal | Paso | Acción |
| | 1 | El usuario selecciona «Coordenadas». |
| | 2 | Se abre la ventana que muestra la imagen con los bordes dibujados. |
| | 3 | El usuario cierra la pestaña. |
| Post-condiciones | Aparece la muestra dibujada. | |
| Excepciones | 1 | No se ha clasificado correctamente el sujeto. |
| Frecuencia | Alta | |
| Importancia | Media | |

Tabla B.4: Caso de uso 4: Mostrar coordenadas

| CU-5: Normalizar | | |
|---|---|---|
| Descripción | Permite al usuario ver la muestra normalizada. | |
| Pre-condiciones | Deberá haberse pulsado «Clasificar». | |
| Requisitos | RF-1, RF-2, RF-2.3 | |
| Secuencia normal | Paso | Acción |
| | 1 | El usuario selecciona «Normalizar». |
| | 2 | Se abre la ventana que muestra la imagen normalizada. |
| | 3 | El usuario cierra la pestaña. |
| Post-condiciones | Aparece la muestra normalizada. | |
| Excepciones | 1 | No se ha clasificado correctamente el sujeto. |
| Frecuencia | Alta | |
| Importancia | Media | |

Tabla B.5: Caso de uso 5: Normalizar

Apéndice $C$

# Especificación de diseño

## C.1. Introducción

Se procederá a explicar la organización de todos los elementos que componen la aplicación.

## C.2. Diseño de datos

### Dataset empleado

Tal y como se explica en la sección 3.7 de la memoria, el dataset empleado para la realización de los experimentos ha sido el *CASIA Iris v1* [1], existían versiones más actuales y grandes, pero se tuvo dificultades para descargarlas, por lo que se optó por la primera versión ya que contenía muestras suficientes como para entrenar un modelo y su peso era de 43 MB.

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 cuentan una resolución de320x280.

Las muestras vienen enumeradas del *001* al *108*, dentro de cada directorio nos encontramos con 2 subdirectorios *1* y *2* que contendrán 3 y 4 muestras respectivamente.

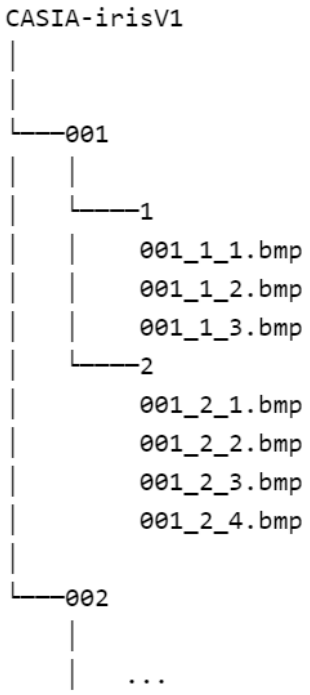


Figura C.1: Estructura de directorios de CASIA Iris V1

Posteriormente se llega a la conclusión de que para el proceso de clasificación se necesitarían etiquetas más claras, ya que establecer que unos atributos extraídos del iris pertenezcan a la clase *025* resulta muy confuso, es por ello que se escogió un dataset de nombres de personas de *Kaggle* y se seleccionaron 108 nombres de dicho dataset para renombrar los directorios de CASIA. De modo que la estructura final del dataset queda así:

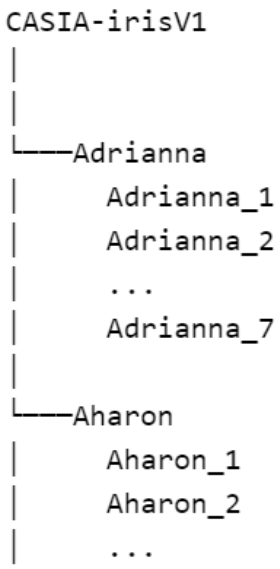


Figura C.2: Estructura de directorios final.

## Datasets generados

A medida que se avanzaba con la experimentación en los *notebooks* se iba generando algunos fichero *.csv* que se necesitarían en etapas posteriores:

- En el *notebook* titulado `#6 Funciones-v2.ipynb` se genera el fichero `iris-data.csv` que contiene las coordenadas (centros y radios) de todas las muestras del dataset. Ver C.2

  Se trataban de las coordenadas correspondientes a los bordes límbico y pupilar que se harían necesarias en la etapa de normalización.

- En el *notebook* titulado `#8 Feature extraction-v2.ipynb` se genera el fichero `iris_features.csv`, que contendrá los atributos extraídos de las muestras polarizadas de 3 modelos de *deep learning* distintos: *VGG16*, *InceptionV3* y *ResNet50*.

## Diagrama de clases

Para el proyecto se crean únicamente 2 clases:

- `Classifier` dentro de `classification.py`: esta clase contiene la lógica para instanciar el modelo de *deep learning* escogido y los métodos para la extracción de atributos y su predicción.
- `Preprocess` dentro de `main.py`:contiene los métodos dedicados a la carga de las muestras que se quieren clasificar, al procesamiento de las muestras y a la posterior clasificación.

  La estructura gráfica y visual de la aplicación también se encuentra contenida en esta clase.

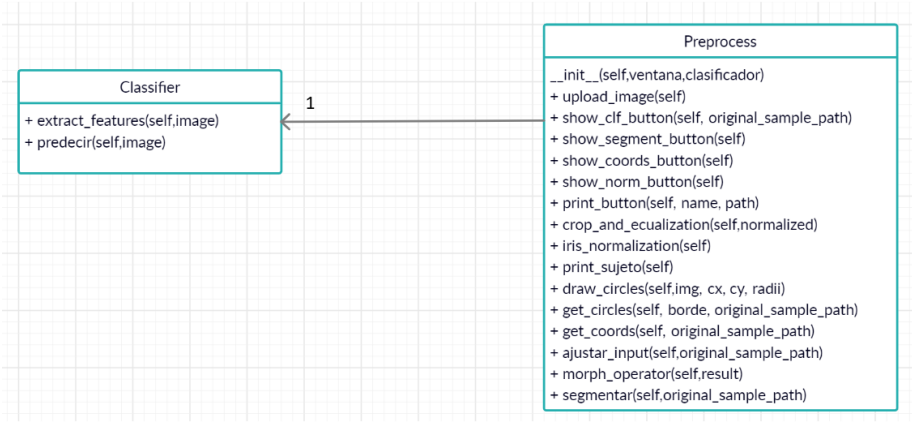


Figura C.3: Diagrama de clases.

## C.3. Diseño procedimental

### Diagramas de secuencia

La aplicación deberá ser capaz de realizar 5 tareas:

- Cargar la muestra que se quiere clasificar C.4.
- Clasificar la muestra C.5.
- Mostrar la muestra segmentada C.6.
- Mostrar las coordendas de la muestra C.6.
- Mostra la muestra noramlizada C.6.

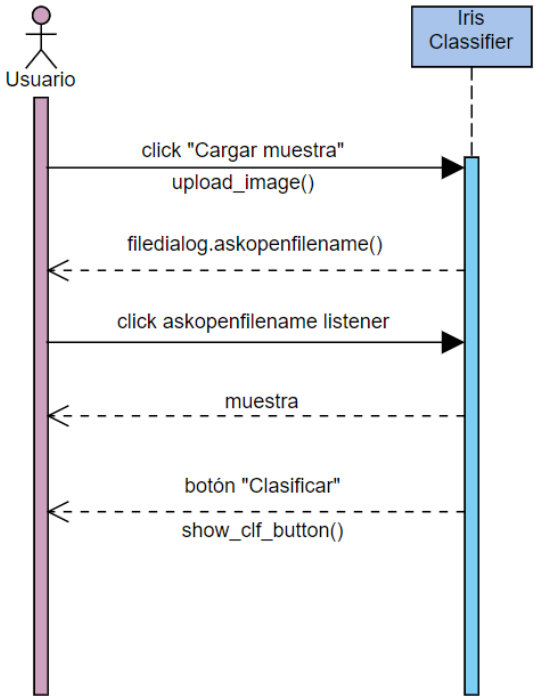


Figura C.4: Diagrama de secuencia para la carga de una muestra.

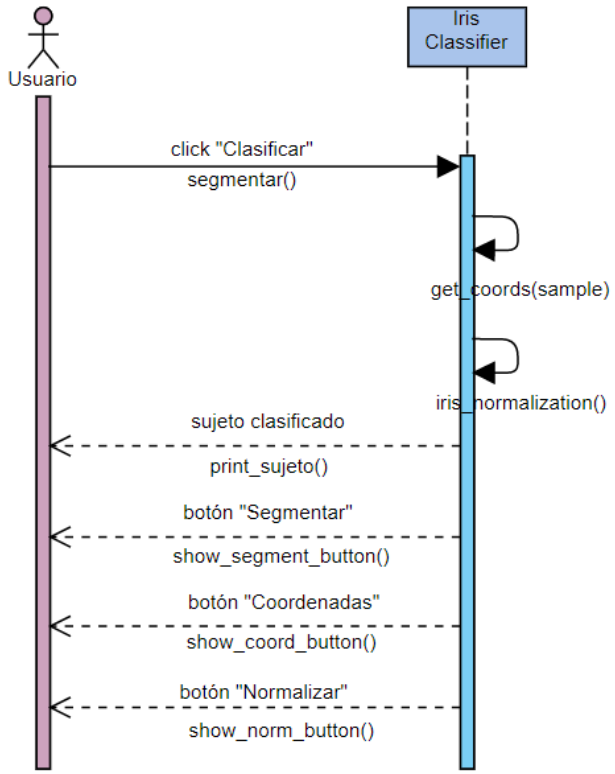


Figura C.5: Diagrama de secuencia para clasificar un sujeto.

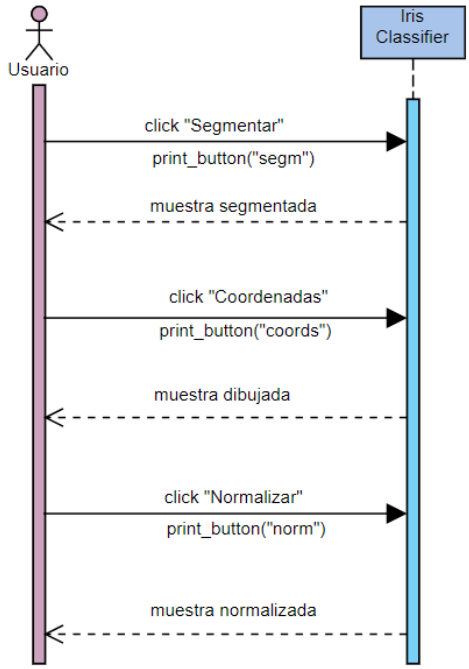


Figura C.6: Diagrama de secuencia para mostrar las imágenes segmentadas, normalizadas y sus coordenadas.

## C.4. Diseño arquitectónico

### Facade Design Pattern

Se intentó seguir el patrón de diseño *facade* [2] en la que se pretende reducir las dependencias entre subsistemas y clientes dividiendo las tareas en clases y proporcionando una interfaz que conecte dichas clases.

Por ejemplo, para este proyecto se crea una clase que engloba las funciones de minería de datos y otra las de procesamiento imágenes, y lo ideal hubiese sido crear una más que permita acoplar las otras dos, pero se optó por usar el *main* en el fichero.*py* que contiene las tareas de procesamiento.

Apéndice $D$

# Documentación técnica de programación

## D.1. Introducción

Se explicará la estructura y organización del proyecto, así como aspectos que se ha de tener en cuenta en caso de que se desee reproducir los resultados obtenidos.

## D.2. Estructura de directorios

Dentro del repositorio encontramos los siguientes ficheros:

- `docs`: contiene la documentación del proyecto.
- `notebooks`: contiene todos los *notebooks* que se han usado para la experimentación así como ficheros generados durante su ejecución.
- `classification.py`: fichero que contiene la clase que se encargará de instanciar el modelo de *deep learning*, de extraer los atributos, y de predecir.
- `main.py`: es el fichero que contiene la aplicación.
- `pickle_model.pkl`: es el clasificador entrenado con los atributos extraídos de los iris.
- `u-net download.md`: red prentrenada usada para la segmentación de las muestras (será necesario descargarla).

- `reqirements.txt`: fichero que contiene las bibliotecas y sus versiones para que la aplicación funcione.

## D.3. Manual del programador

Se detallarán los pasos que ha de seguir otro usuario que desee poner a prueba o mejorar el funcionamiento del sistema.

Si lo que se quiere es probar la aplicación de escritorio directamente se deberá instalar *Python*, para el desarrollo de este proyecto se ha trabajado con la versión *3.8.3* y acto seguido proceder a la instalación de los requisitos necesarios que se encuentran en el fichero `requirements.txt`.

En cambio si se desea explorar el proceso de investigación reflejado en los *notebooks* deberá instalarse además *Anaconda*, aunque de querer ahorrarse este paso se podría usar los cuadernos de *Google Colab*, pero lo más probable es que tarde o temprano surgan problemas que tienen que ver con el acceso a ficheros y directorios.

## D.4. Compilación, instalación y ejecución del proyecto

Pasos para la instalación del proyecto:

- Crear un entorno virtual: `conda create -name iris`
- Cambiarnos a dicho entorno: `conda activate iris`
- Clonar el repositorio del proyecto: https://github.com/jaa0124/iris_classifier
- Instalar los requisitos: `pip install -r requirements.txt`
- Descargar el modelo del fichero `u-net download.md` y alojarlo en la misma raíz del directorio.
- Ejecutar la aplicación: `python main.py`

Para probar los cuadernos simplemente ejecutar *Jupyter Notebook* (que se instala con *Anaconda*) y se podrá empezar a trabajar con ellos y reproducir los resultados obtenidos.

Todos los cuadernos se encuentran en el directorio `notebooks`.

## D.5. Pruebas del sistema

Todos los experimentos realizados se encuentran explicados detalladamente en los cuadernos de *Jupyter*, y los más importantes se han explicado ya en la sección 5.2 y 5.3 de la memoria. Así mismo existen varias versiones mejoradas de los mismos cuadernos que aparecen indicados con el sufijo `-v2`.

Apéndice $E$

# Documentación de usuario

## E.1. Introducción

Se explicará brevemente los requisitos que un usuario normal debe llevar a cabo para ejecutar el proyecto.

## E.2. Requisitos de usuarios

El usuario deberá contar con computador independientemente de si es *Windows* o *Linux* (en mac OS se tiene dudas por ser el único sistema operativo al que no se ha tenido acceso) con *Python* y *Anaconda* instalado.

## E.3. Instalación

Seguir los mismos pasos de la sección D.4.

## E.4. Manual del usuario

Las tareas que permite la aplicación son:

### Cargar muestra

Pasos:

- Pulsar en el botón *Cargar muestra*.

- Seleccionar la muestra deseada.

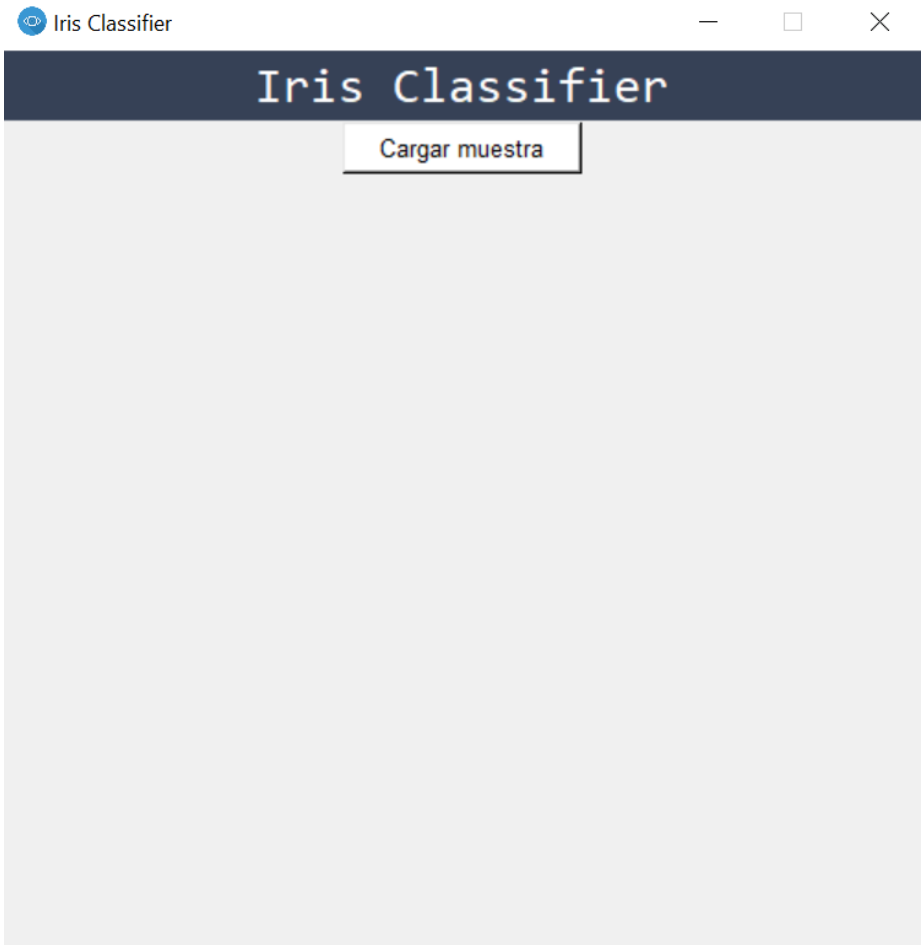


Figura E.1: Pantalla inicial con única opción de cargar una imagen.

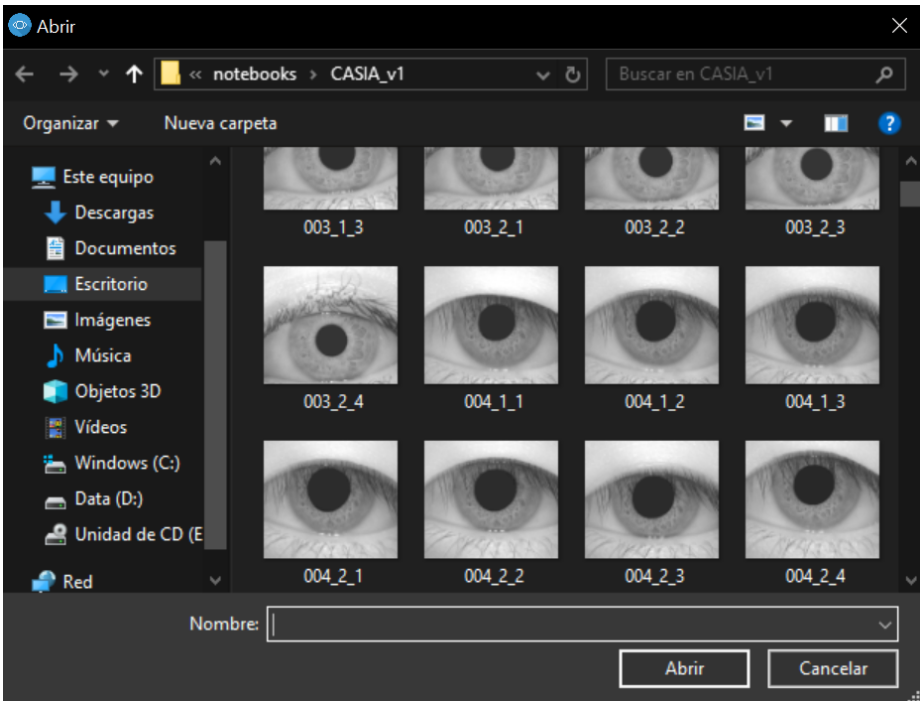


Figura E.2: Ventana de carga.


Figura E.3: Imagen cargado y disponible el botón de clasificar.

## Clasificar

Pulsamos en el botón *Clasificar*.

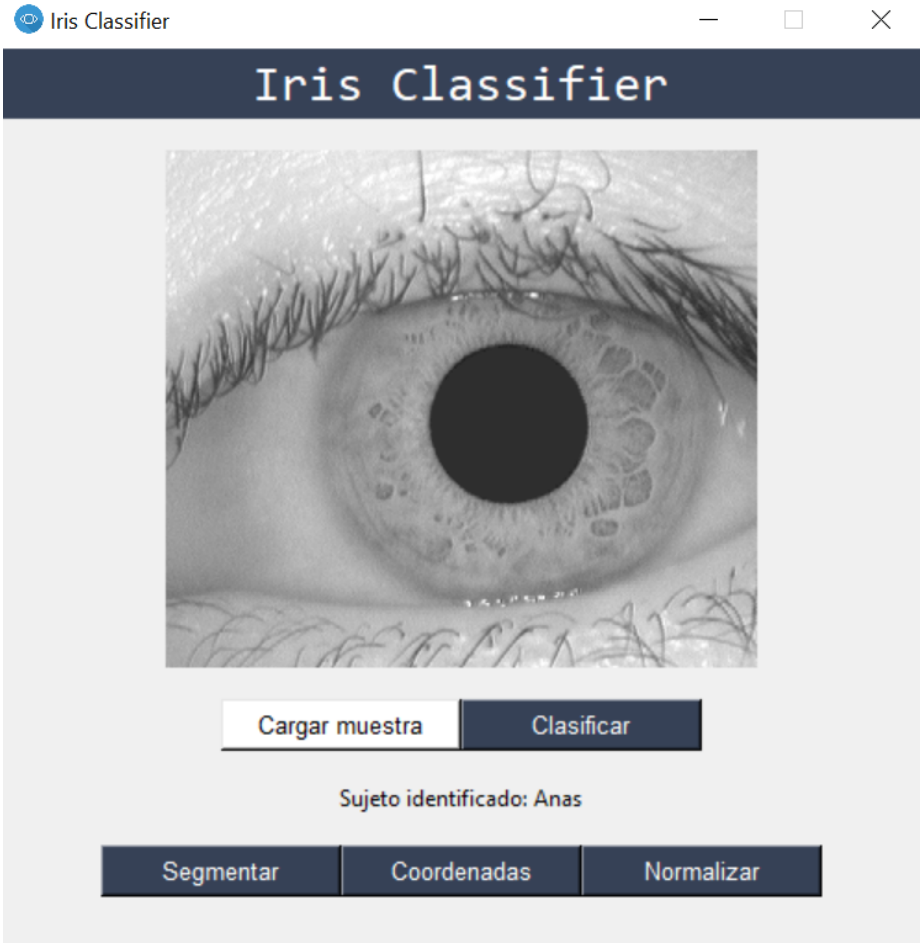


Figura E.4: Resultado de la clasificación.

## Segmentar

Pulsamos en el botón *Segmentar*.


Figura E.5: Muestra segmentada.

## Mostrar bordes límbico y pupilar

Pulsamos en el botón *Coordenadas*.

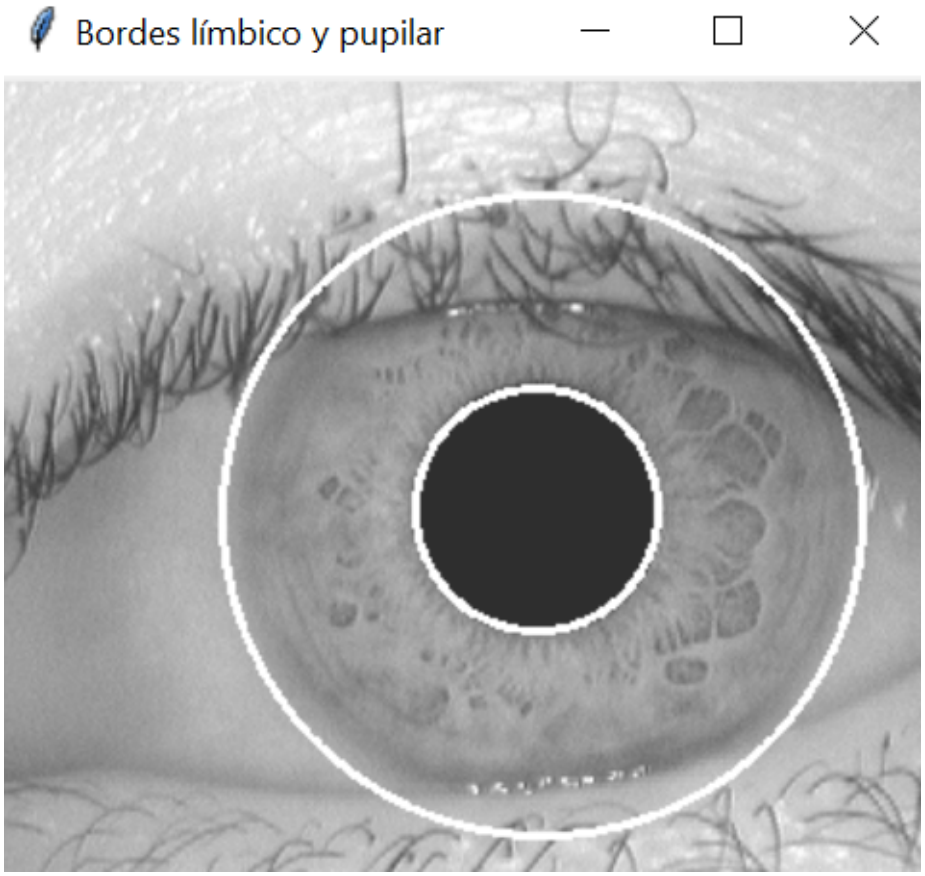


Figura E.6: Muestra con los bordes dibujados.

## Normalizar

Pulsamos en el botón *Normalizar*.

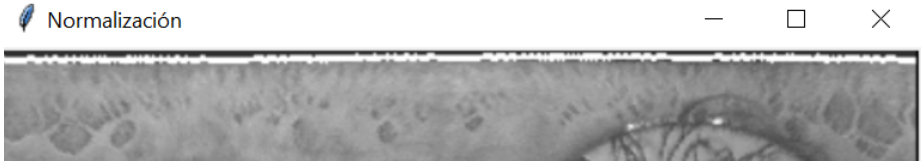


Figura E.7: Muestra normalizada.

Es posible que dependiendo del *hardware* del equipo con el que se trabaje, aparezca el típico aviso de *No responde* después de pulsar el botón *Clasificar*, hacer caso omiso de dicho aviso y en ningún momento cancelar la ejecución.

# Bibliografía

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

[2] Wikipedia. Facade pattern, 2020. [Online; Accedido 27-Junio-2020].