Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Mención Ingeniería del Software

Plataforma basada en redes neuronales para realizar pruebas de autenticación de usuarios mediante datos de ponibles

Autor:

Emilian Vornicu Mihai

Tutor:

Quiliano Isaac Moro Sancho

Índice General

Índice de figuras

Índice de Tablas

Resumen

Durante esta última década la Inteligencia Artificial (IA) y más concretamente el aprendizaje profundo basado en redes neuronales se han convertido en un estándar en varios sectores que afectan diferentes campos de nuestra vida. Estos son capaces de resolver problemas complejos que requieren encontrar patrones ocultos en los datos que tratan.

La identificación es la capacidad de identificar de forma exclusiva a un usuario de un sistema o una aplicación que se está ejecutando en el sistema. Estos modelos tienen la capacidad de identificar a los usuarios mediante los datos ponibles que recogen. Los datos ponibles en nuestro caso hacen referencia a los datos que recogen los relojes inteligentes y mediante los cuales se puede identificar al usuario según sus patrones. Debido a la amplia funcionalidad que ofrecen, es necesario probar ampliamente las distintas configuraciones y capacidades que ofrecen. En este trabajo se propone una plataforma basada en redes neuronales capaz de realizar pruebas de autenticación de usuarios mediante datos de ponibles. En particular, se ha creado un framework o estructura que permite al usuario probar las diferentes funcionalidades que ofrece los modelos de aprendizaje profundo. En concreto, la plataforma permite la selección de dos tipos de modelos, un prototipo de Red Neuronal Artificial (ANN) como es el Perceptrón Multicapa (MLP) y un prototipo de Aprendizaje Profundo basado en Redes Neuronales Convolucionales (CNN).

Palabras clave: Aprendizaje Profundo, Redes Neuronales Convolucionales, Perceptrón Multicapa, Red Neuronal Artificial, framework.

Abstract

During this last decade, Artificial Intelligence (AI) and more specifically deep learning based on neural networks have become a standard in various sectors that affect different fields of our lives. These are capable of solving complex problems that require finding hidden patterns in the data they deal with.

Identification is the ability to uniquely identify a user of a system or an application running on the system. These models have the ability to identify users through the wearable data they collect. Wearable data refers to data that smart watches collect and through which the user can be identified based on their patterns. These models are capable of diagnosing cancer prematurely or proposing better eating habits. Due to the extensive functionality they offer, it is necessary to extensively test the various features and capabilities they offer. In this work, is proposed a platform based on neural networks capable of performing user authentication tests using wearable data. In particular, a has been created that allows the user to test the different functionalities offered by deep learning models. Specifically, the platform allows the selection of two types of models, an Artificial Neural Network (ANN) prototype such as the Multilayer Perceptron (MLP) and a Deep Learning prototype based on Convolutional Neural Networks (CNN).

Keywords: Deep Learning, Convolutional Neural Networks, Multilayer Perceptron, Artificial Neural Network, framewor

Memoria del Proyecto

Capítulo 1
Introducción

En los últimos años, el paradigma informático del aprendizaje profundo (o DL por sus siglas en inglés) se ha considerado el estándar de oro [22] en la comunidad de aprendizaje automático (o ML por sus siglas en inglés). Este, se ha convertido en el enfoque computacional más utilizado en el campo de ML, logrando resultados sobresalientes en varias tareas computacionales complejas. El aprendizaje profundo se ha utilizado recientemente con éxito en muchos campos, como el reconocimiento de objetos visuales, el procesamiento de voz y audio, el procesamiento del lenguaje natural y la detección de objetos, entre otros [6].

La investigación sobre el reconocimiento de actividad basado en relojes inteligentes se ha disparado en los últimos años debido al número creciente de usuarios de estos dispositivos [20]. Se prevé que para el año 2026 el número de usuarios de relojes inteligentes llegue a sobrepasar los 231 millones. Estos dispositivos contienen acelerómetros y giroscopios que detectan los movimientos de un usuario y pueden ayudar a identificar la actividad que está realizando [23]. Aplicando los paradigmas mencionados anteriormente tienen la capacidad de identificar diferentes patrones de comportamiento del usuario, como el sueño, la alimentación o la actividad física y proponer mejoras en estos ámbitos.

Existen diferentes técnicas de Machine Learning que nos permiten resolver el problema de identificación de usuarios [16]. En este trabajo nos enfocaremos en dos modelos diferentes capaces de resolver dicha tarea. Como primer modelo disponemos del Perceptrón Multicapa (o MLP por sus siglas en inglés) que es una Red Neuronal Artificial (o ANN por sus siglas en inglés) también conocido como red neuronal “vainilla” [24]. Como modelo base utilizaremos las redes neuronales convolucionales (o CCNs por sus siglas en inglés) que extiende una red neuronal artificial "clásica" agregando un número muy elevado de capas (aprendizaje profundo), incluyendo la introducción de bloques de convoluciones.

El objetivo del presente Trabajo Final de Grado (en adelante TFG) es proveer una plataforma basada en las redes neuronales que permiten al usuario probar las diferentes funcionalidades que ofrece los modelos de aprendizaje profundo. En concreto, la plataforma permite la selección de dos tipos de modelos, un prototipo de Aprendizaje Profundo basado en Redes Neuronales Convolucionales (CNN) y un prototipo de Red Neuronal Artificial (ANN) como es el Perceptrón Multicapa (MLP) mencionadas con anterioridad. La plataforma permite al usuario seleccionar con relativa libertad los parámetros de estas redes. Además, se permite al usuario procesar los datos en crudo provenientes del smartwatch (giróscopo, acelerómetro y tiempo), y presentar los resultados del modelo ejecutado.

1.1. Objetivos del proyecto

Se detallarán en esta sección los objetivos generales del TFG. En este caso, tiene tanto parte teórica o de estudio, como parte de desarrollo de aplicaciones software. Como objetivos o hitos del presente proyecto se podría contar con:

- Procesar y adaptar los datos ponibles de forma adecuada para tratar el problema.
- Desarrollar un marco de pruebas basado en redes neuronales.
- Construir una aplicación software que permita a los usuarios hacer uso del marco de pruebas creado para redes MLP y CNN.

  Esta aplicación tiene los siguientes requisitos:
  - El usuario debe poder procesar los datos en crudo de los ponibles.
  - El usuario debe poder importar los datos ponibles ya procesados
  - El usuario debe poder seleccionar los parámetros de los modelos
  - La aplicación debe permitir exportar el resultado de la ejecución del modelo
- Este proyecto se debe llevar a un entorno de desarrollo local fácilmente migrable.

1.2. Motivación

En los últimos años, el campo de los sensores, los sensores inteligentes, la inteligencia artificial y la combinación de estos ha progresado enormemente [21]. Específicamente, en el área del aprendizaje automático, las redes neuronales artificiales (ANN), las redes neuronales profundas (DNN) y las redes neuronales convolucionales (CNN), lo que refleja un creciente interés científico y económico en estos campos.

Debido a la amplia funcionalidad que ofrecen, los modelos de IA que realizan dichas tareas están muy refinados y probados hasta obtener un margen de error satisfactorio. Para obtener dichos resultados es necesario probar ampliamente las distintas funciones y capacidades que ofrecen. Esto solo se pude realizar probando las diferentes configuraciones de los modelos, tarea que requiere mucho tiempo. En este trabajo se propone una plataforma basada en redes neuronales capaz de realizar dichas pruebas de autenticación de usuarios mediante datos de ponibles, permitiendo al usuario probar las diferentes funcionalidades que ofrece los modelos de aprendizaje profundo. Esto resuelve el problema mencionado con anterioridad, permitiendo una mayor rapidez en la refinación de dichos modelos.

Conviene señalar que este TFG va a limitarse a trabajar con los datos recogidos y proporcionados por el profesor Dr. Carlos Enrique Vivaracho Pascual. Es, por lo tanto, una herramienta de propósito específico, para la explotación de dichos datos [13].

1.3. Estructura de la memoria

Para la exposición de la memoria se seguirá el siguiente esquema:

- **Capítulo 1. Introducción**. Explicación del contexto, motivación y estructura del documento
- **Capítulo 2. Marco teórico**. Se explican los conceptos teóricos sobre los que se sustenta el trabajo. En primera instancia introduciremos el concepto de las redes neuronales y posteriormente expondremos detalladamente las Redes Neuronales Convolucionales (CNN) y el Perceptrón Multicapa (MLP).
- **Capítulo 3. Marco de trabajo**. Se abordarán las herramientas utilizadas para la resolución del proyecto.
- **Capítulo 4. Metodología**. Se aborda la Metodología de trabajo que se ha seguido en el proyecto.
- **Capítulo 5. Gestión del proyecto**. En este capítulo se analiza la gestión y la planificación del proyecto.
- **Capítulo 6. Datos**. Se analiza la naturaleza de los datos, su recogida y procesamiento.
- **Capítulo 7. Construcción del sistema**. Se analizan las funciones y parámetros que el usuario puede seleccionar de los modelos y la implementación de estos.
- **Capítulo 8**. **Plataforma de trabajo**. Se presenta la GUI de la aplicación con todas sus funcionalidades
- **Capítulo. 9 conclusiones**. En este apartado se analizan las conclusiones de este proyecto y las futuras aplicaciones y mejoras de este proyecto.
- **Manual de instalación y guía de uso.** En este apartado se analizarán los pasos a realizar para ejecutar el proyecto.

Capítulo 2
Marco Teórico

Una red neuronal artificial es modelo computacional formado por un grupo interconectado de nodos o neuronas artificiales capaces de producir un valor de salida que den resultado a un problema planteado. Estos intentan imitar el funcionamiento de la vasta red de neuronas de un cerebro biológico [10].

Los primeros que introdujeron el concepto de red neuronal fueron Warren McCulloch y Walter Pitts (1943) creando un modelo informático para redes neuronales [25]. Este modelo intentaba imitar el comportamiento de una neurona biológica y la forma de cálculo de estas. Para entender el funcionamiento de las redes neuronales artificiales nos fijaremos en cómo funcionan las neuronas biológicas.

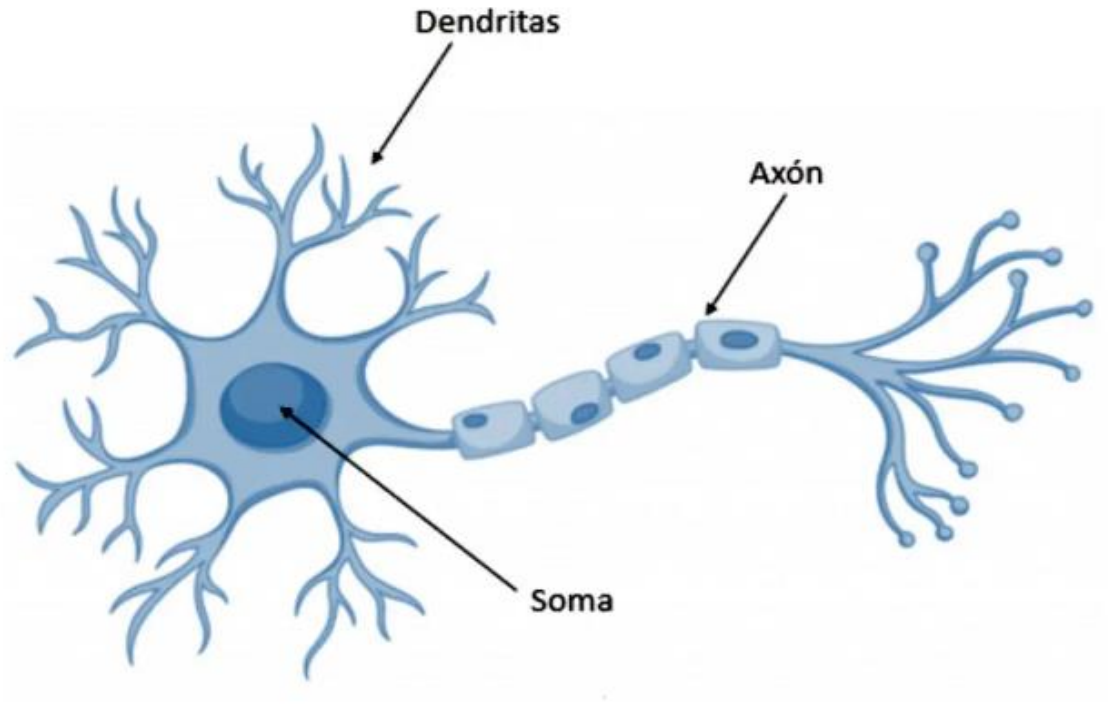


Figura 1 Representacion de una neurona biológica

Una neurona biología está formada por dendritas, el soma y el axón. Las dendritas son las que captan los estímulos externos que llegan a la neurona, estos se procesan en el Soma o núcleo de la neurona y finalmente se envían a otras neuronas mediante el Axón formando así una red [3].

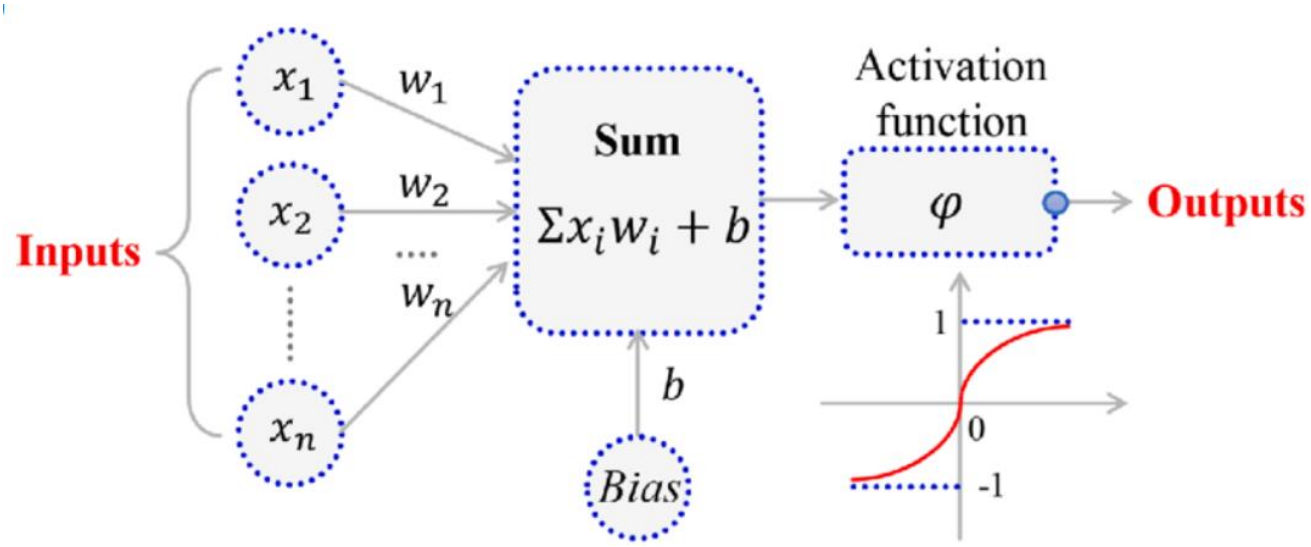


Figura 2 Similitud entre neurona artificial y neurona biológica

Las Neuronas artificiales funcionan de la misma manera en el modelo planteado por McCulloch y Pitts [25], tal como se observa en la figura…. Estas reciben unos Inputs o "estímulos" que son $x_1, x_2, \dots x_n$ simulando así la función biológica de las dendritas. Cada input o entrada tiene asociada un peso $w_1, w_2, \dots w_n$, que se emplearía para hacer la sinapsis o intercambio de información. El núcleo o soma viene representado mediante el sumatorio de sinapsis y el umbral de activación o Bias, al que posteriormente se le aplica una función de activación. La función de activación φ , emula el proceso que realiza el cuello del axón. La fórmula expuesta en el modelo es la siguiente:

$$z = \sum_{j=1}^{n} (x_i . w_j) + b$$

$$y = \varphi\,(z)$$

El modelo planteado McCulloch y Pitts [25] se basa en propagación hacia delante, es decir a partir de unos pesos asignados a las conexiones entre neuronas y unos datos iniciales, se calcula la salida de la red. En los años siguientes, más concretamente en el año 1974 Werbos [26] diseñó la propagación para atrás o del inglés backpropagation, en el que la salida de neuronas sirve de entrada para los nodos siguientes, esta permite el reajuste de pesos para obtener una mejor solución. La función de activación mencionada con anterioridad es una función matemática aplicada a la salida de la neurona para modificar y añadir deformaciones no lineales, su impacto contribuye al rendimiento de la red modelando el resultado de esta [1].

Hay diferentes arquitecturas y topologías de redes neuronales, cada una tiene funcionalidades diferentes. Por ejemplo, existen Redes Neuronales Autoorganizadas, Redes Neuronales Recurrentes, Redes de Base Radial …etc.

A continuación, abordaremos el Perceptrón Multicapa (MLP) y más concretamente las Redes Neuronales Convolucionales (CNN).

2.1. Perceptrón Multicapa (MLP)

La idea de perceptrón multicapa nace a raíz del descubrimiento realizado en 1969, Minsky y Papert [18], que demuestran que el perceptrón simple y ADALINE [28] no puede resolver problemas no lineales como sucede con las XOR. Se sabía que la combinación de varios perceptrones simples podría resolver ciertos problemas no lineales como el mencionado con anterioridad, pero no existía un mecanismo automático para adaptar los pesos de la capa oculta.

No fue hasta el año 1986 cuando Rumelhart y otros autores, presentan la "Regla Delta Generalizada" que permite la propagación hacia atrás de los errores que permitiendo adaptar los pesos [4]. De esta forma se consigue trabajar con múltiples capas y con funciones de activación no lineales. Se demuestra que el perceptrón multicapa es un aproximador universal. Un perceptrón multicapa puede aproximar relaciones no lineales entre los datos de entrada y salida. Debido a su utilidad esta red se ha convertido en una de las arquitecturas más utilizadas hasta el momento.

Un perceptrón multicapa o MLP (por sus siglas en inglés MultiLayer Perceptrón) es aquel formado por varias capas de neuronas, en concreto una capa de entrada, una capa de salida, y como mínimo una capa intermedia a la que denominamos capa oculta.

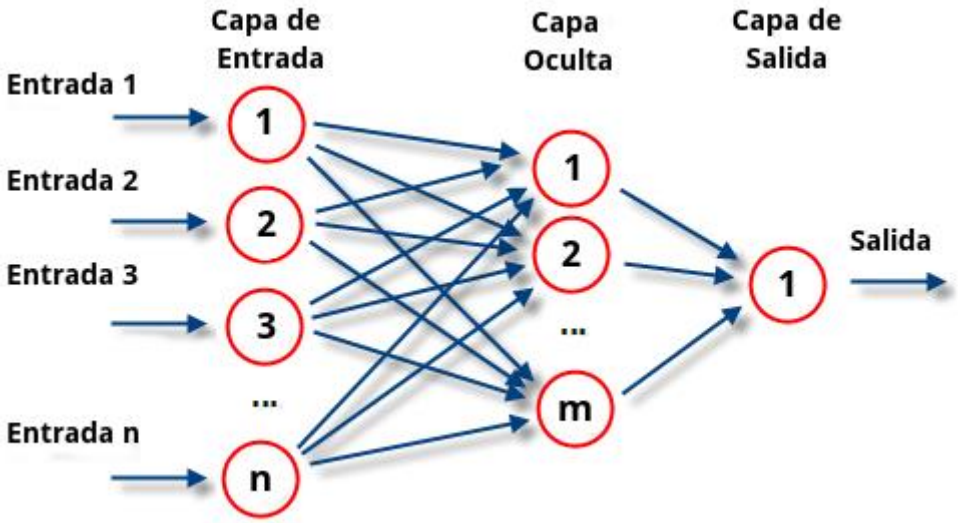


Figura 3 Representacion de un modelo MLP

Cada neurona en esta capa representa un atributo o propiedad del ejemplo a clasificar. La predicción se realiza a partir de los valores del conjunto de atributos que describe el ejemplo.

El número de capas ocultas también es una elección en el diseño de la red. No obstante, hay que tener en cuenta que añadir muchas capas ocultas, puede no beneficiar al aprendizaje y aumentar el coste computacional, resultando en un sistema innecesariamente más lento. En un problema de tamaño pequeño no es necesario aumentar el número de capas ocultas, sin embargo, si el tamaño de los datos a tratar es mayor,

por ejemplo, para tratar imágenes, sí que está demostrado que aumentar M es necesario, ya que se obtienen mejores resultados.

Todas las neuronas están conectadas con las neuronas de la siguiente capa a excepción de la neurona bias de dicha capa. Por lo tanto, la función de activación de una neurona depende de la entrada que reciba, es decir de todas las neuronas de la capa anterior.

El aprendizaje en el perceptrón multicapa se produce al actualizar los pesos de las neuronas después de procesar cada dato, en función de la tasa de error que compara el error de la salida con el resultado esperado. A continuación, se muestra con un ejemplo matemático como se realiza este aprendizaje mediante backpropagation que es que es una generalización del algoritmo de mínimos cuadrados medios (LMS) [2].

A continuación, se representa el error de un nodo de salida de un conjunto de entrenamiento mediante la siguiente función:

$$e_j(n) = d_j(n) - y_j(n)$$

Donde *d* es el valor objetivo, e *y* es el valor producido por el perceptrón.

Los pesos de los nodos buscan minimizar el error de salida utilizando la siguiente función:

$$\varepsilon(n) = \frac{1}{2}\sum_j e_j^2(n)$$

Usando el gradiente descendiente, el cambio que se produce en cada peso es el siguiente:

$$\Delta w_{ji}(n) = -\eta \frac{\partial \varepsilon(n)}{\partial v_j(n)} y_i(n)$$

Donde y_i es la salida de la neurona anterior y η es la tasa de aprendizaje del modelo.
La deriva a calcular depende del $v_j(n)$ que no varía, por tanto, podemos simplificar la derivada a la siguiente expresión:

$$-\frac{\partial \varepsilon(n)}{\partial v_j(n)} = e_j(n)\ \emptyset'(v_j(n))$$

Donde $\emptyset'$ es la derivada de la función de activación que no varía, por tanto, la derivada relevante es la es la siguiente:

$$-\frac{\partial \varepsilon(n)}{\partial v_j(n)} = \emptyset'\left(v_j(n)\right)\sum\nolimits_k -\frac{\partial \varepsilon(n)}{\partial v_j(n)}\, w_{kj}(n)$$

Esta derivada depende del cambio de los pesos de los k nodos, que representa la capa de salida. Por tanto, para cambiar los pesos de la capa oculta, los pesos de la capa de salida cambian de acuerdo con la

derivada de la función de activación, por lo que este algoritmo representa una propagación hacia atrás de la función de activación [11].

2.2. Perceptrón Multicapa en la práctica

En la práctica para la implementación del modelo MLP utilizaremos la clase MLPClassifier de Sklearn. Como el objetivo de este proyecto es crear una plataforma de pruebas para modelos de IA, debemos comprender los pasibles parámetros de las redes y para que se utilizan. La clase MLPClassifier permite seleccionar distintos parámetros del modelo MLP, los más importantes son los siguientes:

- Hidden_layer: número de capas ocultas que tiene el modelo. Cada capa oculta tiene por defecto 100 neuronas.
- Activation: permite la selección de la función de activación de las capas ocultas, tenemos las siguientes opciones:
    - *Indentity*: no utiliza la función de activación, devuelve f(x) = x
    - *Logistic*: utiliza la función sigmoide cono activación
    - *Tanh*: utiliza la funcion hiperblica como activacion
    - *Relu*: devuelve el máximo entre 0 y el valor, este activador es el que se usa por defecto.
- solver: se utiliza para la optimización de pesos, tenemos las siguientes opciones:
    - *lbfgs*: es un optimizador perteneciente a la familia de los métodos cuasi-Newton
    - *sgd*: hace referencia al descenso de gradiente estocástico
    - *Adam*: hace referencia al gradiente estocástico propuesto por Kingma, Diederik, y Jimmy Ba
- batch_size: permite seleccionar el tamaño de baches más pequeños para los optimizadores estocásticos. Por defecto tiene el valor de automático
- learning_rate: tasas de aprendizaje del modelo, tenemos las siguientes opciones:
    - *constant*: tasa de aprendizaje constante, vienen dada por *learning_rate_init,* es el valor por defecto
    - *invscaling*: tasa de aprendizaje inversa, gradualmente va descendiendo la tasa de aprendizaje a lo largo de ejecuciones
    - *adaptive*: tasa de aprendizaje adaptativo a medida que la tasa de error del entrenamiento va disminuyendo.
- Max_iter: máximo numero de iteraciones del modelo hasta que converge. Por defecto este valor está establecido a 200.

2.3. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales son un tipo especial de Red Neuronal Densa o Perceptrón Multicapa, ya que todas las neuronas que componen la red están conectadas entre sí.

La base de las CNN tiene inicio en el año 1959, con el descubrimiento de dos tipos de células distintas en el córtex primario, se descubrió la célula responsable de la orientación y la célula responsable de los estímulos visuales. Hubel y Wiesel responsables de este hallazgo determinaron que las neuronas del córtex visual son sensibles a la posición y a la orientación [5].

No fue hasta los inicios de los años 80 cuando se establecieron las bases de las Redes neuronales Convolucionales con la primera red neuronal de este tipo, denominada Neocognitron por parte de Kunihiko Fukushima [15] que se basó en el hallazgo de Hubel y Wiesel. La combinación de células o neuronas de posición y orientación en un modelo en cascada, permitía reconocer patrones visuales complejos mediante una unión de sub-componentes o componentes de menor nivel, que habían sido previamente reconocidos por el modelo.

Finalmente, en el año 1998, Yann LeCun mejoró el modelo creado por Kunihiko Fukushima introduciendo el backpropagation o la propagación hacia atrás [27]. Este modelo mejora aumento el rendimiento del modelo anterior y permitió la actualización de pesos automática. El uso principal de las Redes Neuronales Convolucionales se da en el campo de las imágenes siendo consideradas un referente en la clasificación y tratamiento de estas.

La diferencia fundamental entre las Redes neuronales densas como puede ser Perceptrón Multicapa (MLP) mencionado con anterioridad y una Red convolucional (CNN) es el patrón de aprendizaje. En las primeras redes, los patrones que se aprenden son globales mientras que en las redes neuronales convolucionales los patrones que se aprenden son locales o por zonas si nos referimos a imágenes [12].

Por ejemplo, una primera capa de aprender patrones en un lado de la imagen otra capa aprende patrones más grandes que contienen las zonas analizadas en las primeras capas y así en adelante. Esto permite a estas redes aprender patrones abstractos eficazmente.

Otra característica que poseen es que, los patrones que se aprenden las CNN son invariables a la traslación, una vez aprendidos se pueden reconocer en cualquier otro lado, por ejemplo, una vez aprendido el patrón de una zona de la imagen, el modelo es capaz de reconocerlo en cualquier otro ámbito. Mientras que el modelo MLP tendría que aprender de nuevo el patrón si se lo vuelve a encontrar.

Otra diferencia que incorporan las CNN es la operación matemática de Convolución de la cual reciben el nombre y que se explicará a continuación.

2.3.1. Operación Convolución

La operación de convolución permite expresar, la relacion existente entre dos funciones con argumento real, es una operación que transforma dos funciones, en una tercera. Expresado matemáticamente la operación de convolución se representa de la siguiente manera:

$$H(x) = \int_{-\infty}^{+\infty} I(z)K(x-z)\,dz$$

Una versión discretizada de la expresión continúa expuesta anteriormente, es la siguiente:

$$H(x) = \sum_{m=\infty}^{\infty} I(m)K(x-m)$$

En nuestro caso, I hace referencia a los datos de entrada que puede ser un array de varias dimensiones. K hace referencia al kernel, que es un array multidimensional al igual que I, que contiene los parámetros de los que deseamos aprender los valores. Normalmente el tamaño de K es mayor que el de I. Y por último H es la salida de la operación de convolución, denominada mapa de características o feature map.

Para poder realizar una operación de convolución, se tienen que determinar que I y K valgan cero en todas las posiciones, excepto en aquellas que se corresponden con el dominio de interés de nuestro problema.

Por tanto, la función ya está acotada al dominio deseado. En un caso 2D que es el más común en estas redes para poder emplear la convolución, es necesario interpretar I y K como un array bidimensional, en el caso de que fuera 3D estos serían arrays de tres dimensiones. En este caso, K puede ser considerado también de dos dimensiones. Obtenemos la siguiente expresión:

$$H(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Al aplicar la propiedad conmutativa, obtenemos una formulación más óptima para la resolución de estos problemas:

$$H(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Cuando el Kernel es de menor tamaño que los datos de entrada I que es el caso más habitual, para realizar la convolución, se interpreta el Kernel como una ventana móvil que se va desplazando a lo largo de los

datos de entrada, realizando las operaciones de convolución. Esta idea puede verse claramente en la *Figura 4*, en la que se toma como datos de entrada o input un array bi-dimensional.

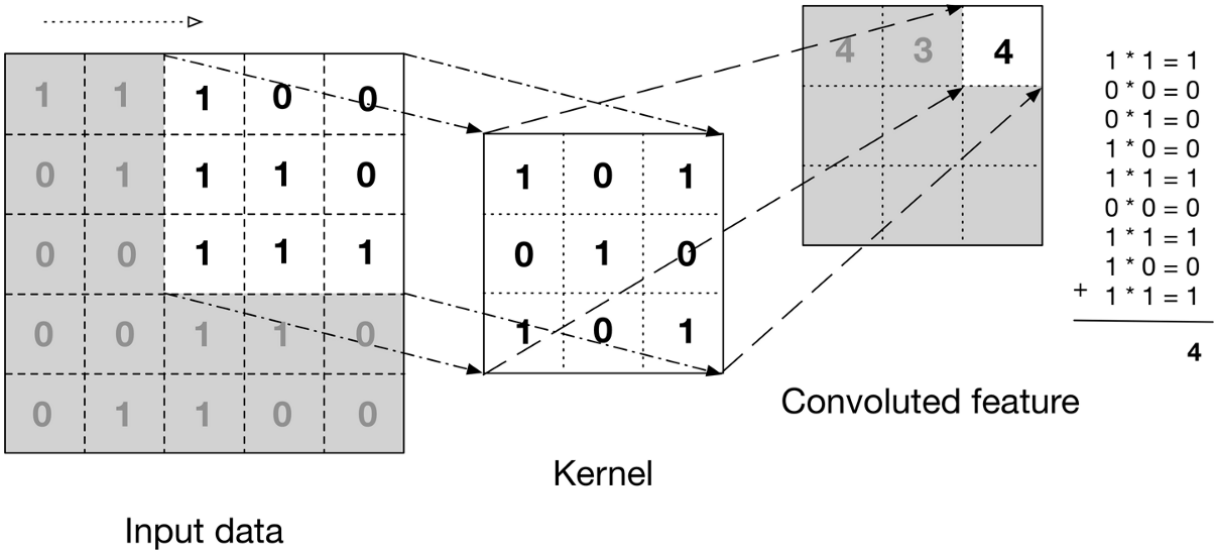


Figura 4 Ejemplo de convolucion 2D

Tal como se observa en la *Figura 4* el kernel se desplaza siguiendo un movimiento semejante a la lectura, es decir de izquierda a derecha y de arriba abajo. Para cuantificar el desplazamiento entre una aplicación del kernel y la siguiente tenemos el concepto de strides. El stride es un parámetro que modifica la cantidad de movimiento sobre la imagen o el vídeo. Por ejemplo, si el stride de una red neuronal se establece en 2, tal como se observa en la *figura 5* el filtro se moverá dos píxeles, o dos unidades, cada vez, ya sea movimiento en filas o en columnas. El tamaño del filtro afecta al volumen de salida codificado, por lo que el stride suele establecerse en un número entero

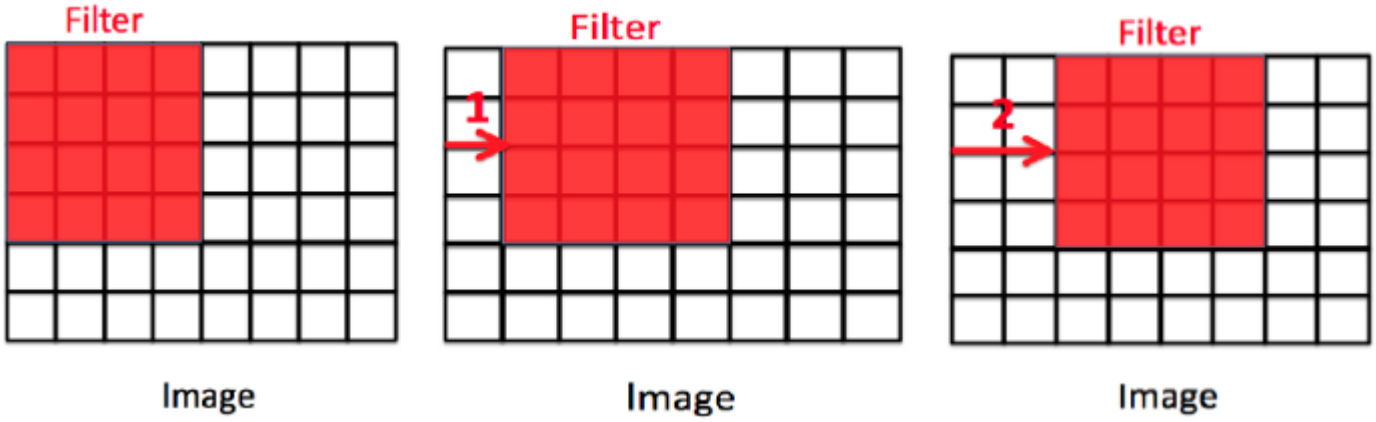


Figura 5 Ejemplo de funcionamiento del parametro stride

Otro detalle importante de la *figura 5* es que el mapa característico resultante o feture map es de menor tamaño que los datos de entrada o input data. Este feature map sirve de entrada para otras neuronas y así sucesivamente, esto provoca que a lo largo de las capas se reduzca la dimensionalidad.

Para evitar esto se utiliza el padding o rellenado. Tal como su nombre indica consiste en añadir pixeles en los bordes. Mas concretamente, consiste en añadir las filas y columnas necesarias a cada lado del map

feature que sirve de entrada de una convolución. De esta forma, lo que se consigue es que la salida tenga el mismo tamaño o dimensión que la entrada.

Con respecto al padding podemos tener tres casos:

- Zero-padding o same convolution: tal como su nombre indica es técnica que añade pixeles de 0, en filas y columnas tal como se observa en *la figura 6* aumentando la dimensión de la imagen, de esta forma no se reduce la dimensión de salida después de la convolución. El inconveniente de esta técnica es que se disminuye la influencia de los pixeles cercanos al borde.

- Valid convolution: no se realiza padding, lo que disminuye el mapa de características al realizar la operación de convolución

- Full padding o full convolution: se añaden tantos ceros o pixeles como sea necesario para que cada pixeel sea visitado n veces por el kernel. Esta técnica solventa el problema de una mayor influencia de unos pixeles con respecto a otros.

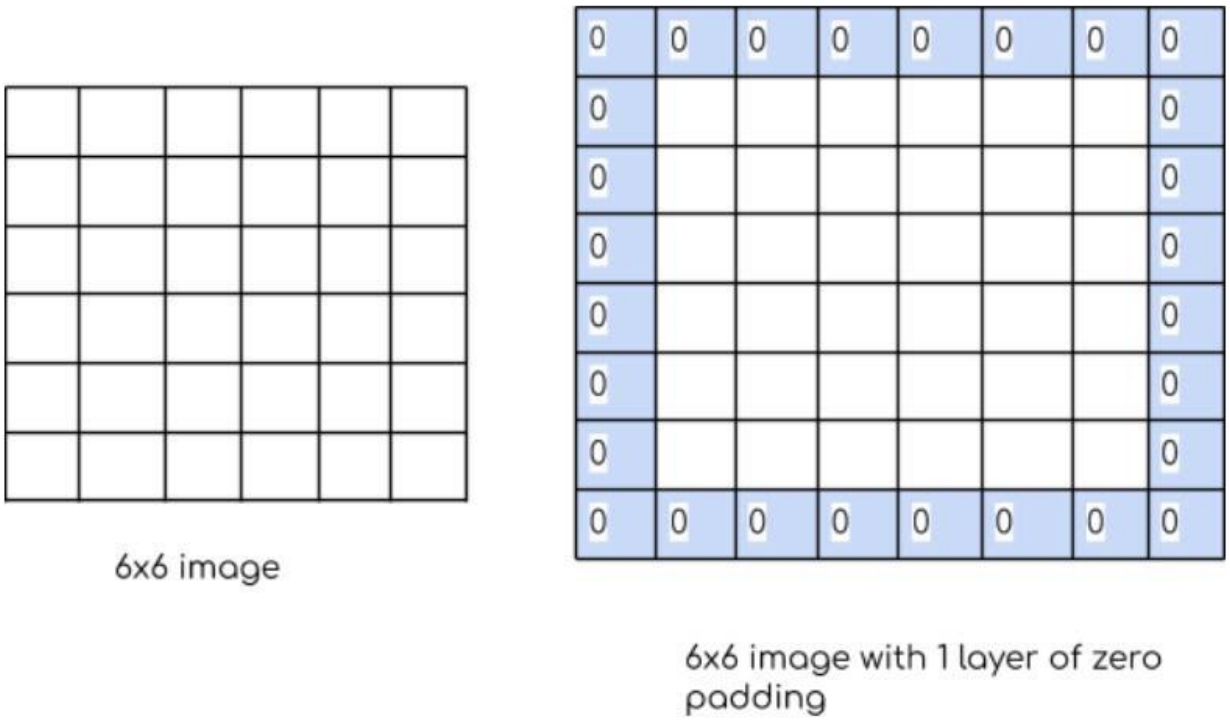


Figura 6 Ejemplo de Zero padding

En comparación a las redes neuronales que utilizan el multiplicado tradicional de matrices la operación de convolución ofrece las siguientes ventajas:

- Conectividad dispersa: En contraste a lo que ocurre con las redes neuronales tradicionales, en las redes neuronales convolucionales, no es necesario que las neuronas de una capa tengan como entrada las salidas de todos los nodos de la capa previa. Solo un grupo pequeño de neuronas se activan en un cierto momento. Esto se consigue dando dimensiones del kernel más pequeño, que las de entrada. Por ejemplo, al procesar una imagen de entrada, que tiene una gran cantidad de pixeles, miles o millones, podemos detectar características pequeñas y significativas, como los

bordes, con núcleos que sólo ocupan decenas o cientos de píxeles. Esto significa que necesitamos almacenar menos parámetros, lo que reduce los requisitos de memoria del modelo y mejora su eficiencia estadística. También el cálculo de su salida requiere de menos operaciones.

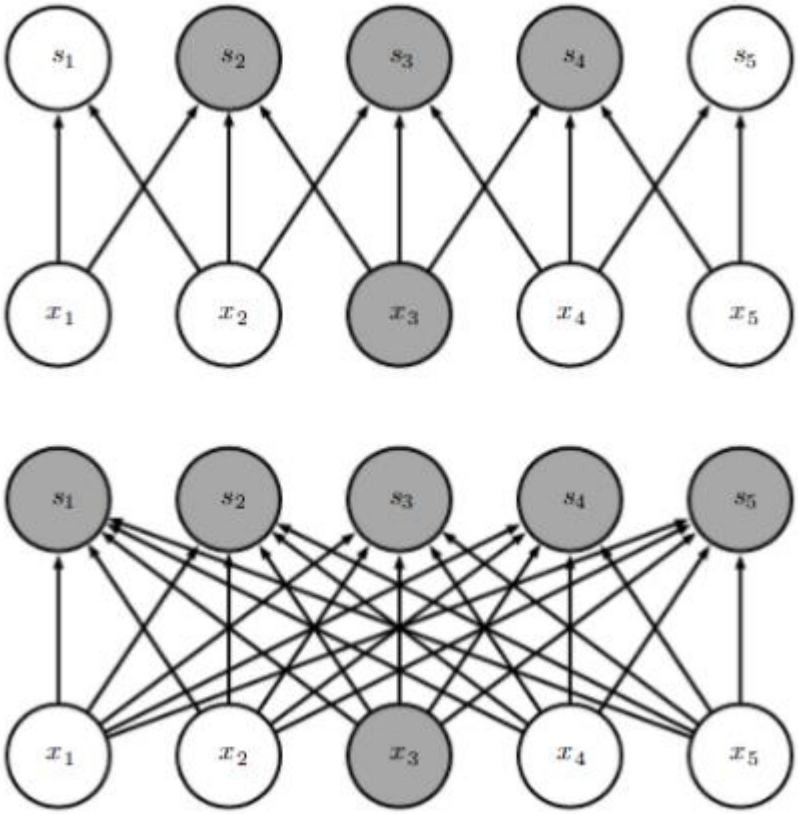


Figura 7 Ejemplo de red con conectividad densa y red con conectividad dispersa

- Representaciones equivalentes: Debido a que los parámetros son compartidos, se da la propiedad de equivalencia dentro de la capa, por tanto, el impacto de una traslación en la entrada no afecta a la salida. Sin embargo, no es equivalente a otros cambios, como la rotación y el escalado. Son necesarios otros métodos para manejar este tipo de transformaciones.

- Compartición de parámetros: En una red neuronal habitual, MLP, cada parámetro se usa exactamente una vez para calcular la salida de una neurona de una capa. En cambio, en las redes convolucionales cada peso del Kernel está ligado es decir se emplea varias veces, en cada operación de convolución. Esto tiene como consecuencia un aprendizaje basado en un único conjunto de datos

Otra de las operaciones fundamentales de las Redes Neuronales Convolucionales es el pooling o submuestreo o agrupación de características.

2.3.2. Pooling

Un problema importante con las capas convolucionales es que el mapa de características producido por el filtro depende de la ubicación. Esto significa que, durante el entrenamiento, las redes neuronales convolucionales aprenden a asociar la presencia de una determinada característica con una ubicación específica

en la imagen de entrada. Esto puede reducir gravemente el rendimiento. En su lugar, queremos que el mapa de características y la red sean invariantes a la traducción. La operación pooling ayuda a que la representación sea aproximadamente invariable frente a pequeñas traslaciones de la entrada.

Podemos destacar diferentes tipos de pooling, los más populares son:

- AveragePooling: submuestrea la entrada a lo largo de sus dimensiones espaciales, altura y anchura, tomando la media como valor de salida en la ventana de entrada

- Max pooling: igual que el anterior, pero la operación empleada es el cálculo del máximo, en vez de la media

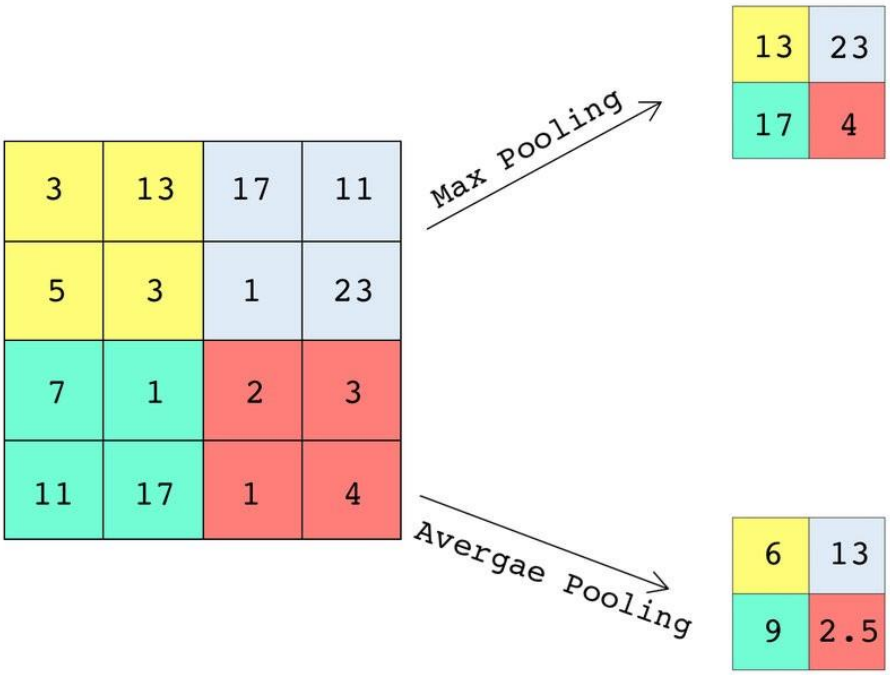


Figura 8 Ejemplo de Max Pooling y de Average Pooling

- GlobalAveragePooling: en esencia es igual que AveragePooling pero ya tiene definido el tamaño de la ventana, que coincide con el tamaño del mapa de características.

- GlobalMaxPooling: en esencia, es igual que MaxPooling, pero ya tiene definido el tamaño de la ventana, que coincide con el tamaño del mapa de características

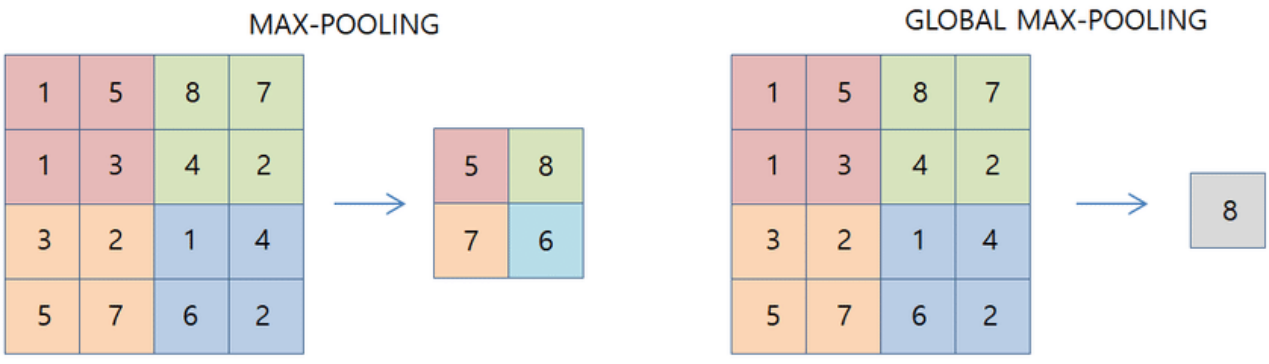


Figura 9 Ejemplo de max pooling y de global max poling

La elección de uno o de otro depende de los objetivos que se persigan, en cualquier caso, el pooling ayuda a que la representación sea aproximadamente invariable a pequeñas traslaciones de la entrada. Es decir, si la entrada se traslada en una cantidad pequeña, los valores de las salidas no cambian.

2.3.3. Dimensiones

Las redes Convolucionales no solo se utilizan para el tratamiento de imágenes, también se utilizan para datos temporales, tratamiento de modelos 3D y así en adelante.

Según el número de dimensiones que tengan los datos de entrada la arquitectura, la estructura de datos o la naturaleza del problema a tratar estas pueden ser:

- Unidimensionales(1D): para datos temporales como los vectores. Este es nuestro caso, utilizaremos datos ponibles recibidos de un reloj inteligente.

- Bidimensionales (2D): para datos espaciales, matrices. Se centra principalmente en el análisis de imágenes

- Tridimensionales (3D): para datos espaciales-temporales. En este caso hablamos de una mezcla de las dos estructuras anteriores, un caso de esto es el tratamiento de vídeo, que es una secuencia de imágenes a lo largo del tiempo.

2.3.4. Redes CNN en la práctica

En la práctica para la implementación del modelo CNN utilizaremos Keras de Tensor Flow. Como el objetivo de este proyecto es crear una plataforma de pruebas para modelos de IA, debemos comprender los pasibles parámetros de las redes y para que se utilizan. Como los modelos CNN se organizan en capas, las clases que utilizaremos para la implementación de las capas del modelo CNN son las siguientes:

De cada capa se mostrarán los parámetros más importantes.

- **Conv1D**
  - Filters: indica el número de dimensión de la capa.
  - Kernel_size: numero o una tupla que indica el tamaño de la ventana convolucional 1D.
  - Strides: número o tupla que especifica el valor de strides. Si no se especifica, por defecto tiene el valor 1.
  - Padding: Hay tres opciones diferentes. Si no se especifica por defecto Valid.
    - Valid: significa que no se aplicara padding.

        - Same: se añade padding con 0s hasta que el input tiene la misma salida que el output

        - Causal: se utiliza para modelos que usan datos temporales, evita la dilatación

    - Activation: Función de activación a utilizar. Por defecto no tiene valor asociado

- **MaxPooling1D**
    - Pool_size: indica el tamaño de la ventana MaxPooling
    - Padding: Hay dos opciones diferentes. Si no se especifica por defecto Valid.

        - Valid: significa que no se aplicara padding.

        - Same: se añade padding con 0s hasta que el input tiene la misma salida que el output

- **Dropout**
    - Rate: numero decimal entre 0 y 1, indica la fracción del input a la que hacer el drop.
    - Seed: numero entero de Pyton que se usa como semilla aleatoria

- **Flatten**
    - Data: datos a los que transformar en matriz unidimensional

- **Dense**
    - Units: Numero que indica la dimensión de la capa dense
    - Activation: función de activacio. Si no se especifica nada, el valor por defecto es “linear”

Aunque no es una capa sino una opción del modelo, se analizara también los parámetros más importantes de la función compile que sirve para compilar el modelo.

- **Compile**
    - Optimizer: parámetro que permite elegir el optimizador de compilación. Si no se especifica nada por defecto tiene el valor *RMSprop*
    - Loss: parámetro que permite seleccionar la función de pérdida del modelo.

Capítulo 3

Marco de trabajo

Un marco de trabajo para la gestión de proyectos consta de los procesos, las tareas y las herramientas que se utilizan para llevar a cabo un proyecto de principio a fin. Abarca todos los componentes clave necesarios para planificar, gestionar y dirigir proyectos [19].

En esta sección encontraremos las tecnologías que se han empleado para el desarrollo del proyecto y la aplicación.

3.1. Hardware

Para la elaboración del proyecto se ha utilizado un ordenador personal, con un i5-6600k, 16 Gb de RAM, 250 Gb de disco SSD y 2 Tb de disco duro sólido. A continuación, se muestran las características del ordenador utilizado para el desarrollo del proyecto:

Figura 10 Especificaciones del dispositivo

3.2. Software

Para el desarrollo y documentación de este proyecto se han utilizado distintas aplicaciones Software que se mencionaran a continuación:

- *Cuadernos Jupyter*: Para una fase inicial se utilizaron cuadernos de Jupyther, pero estos solo se podían utilizar en local lo que imposibilitaba la programación de cualquier lugar que no fuera el ordenador personal, también resultaba complicado realizar entregas regulares debido a que siempre se tenía que enviar el fichero[14].

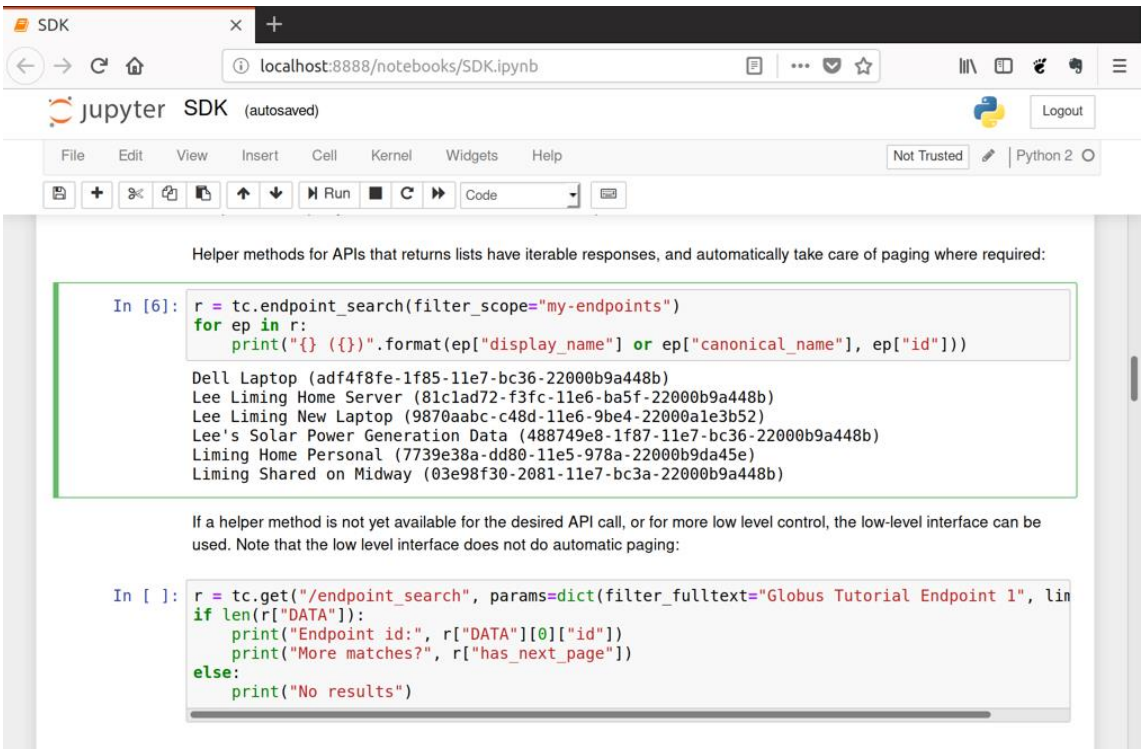


Figura 11 Cuaderno jupyter

- *Google Colab*: Para resolver los problemas mencionados con anterioridad, se ha optado por programar en Google Colab, que permite escribir y ejecutar código de Python en la nube. Es especialmente adecuado para tareas de aprendizaje automático, análisis de datos y educación [8]. Esto tiene la ventaja de que el cliente siempre puede ver en qué estado se encuentra el proyecto y puede escribir comentarios para dar retroalimentación al programador. También nos ha servido de repositorio hasta el sprint 3.

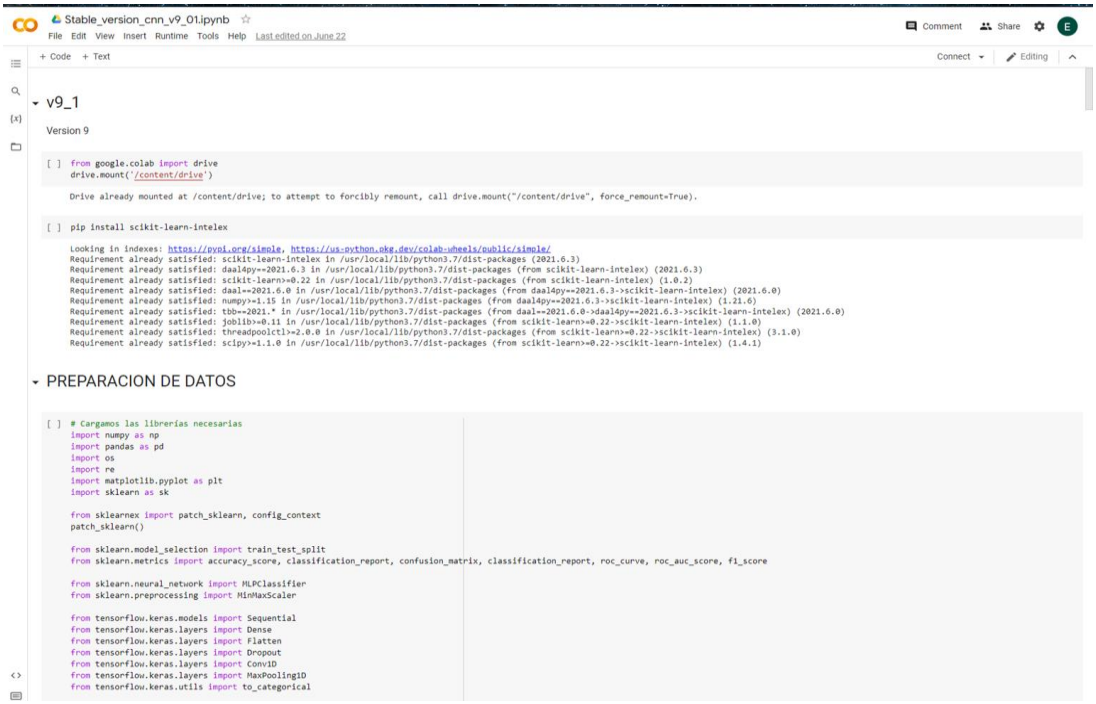


Figura 12 Plataforma Google Colab

- *Google Drive*: Hemos utilizado Google drive como repositorio para ir subiendo las versiones iniciales y los cuadernos Júpiter durante las primeras fases, y por último lo hemos utilizado para subir el código del proyecto

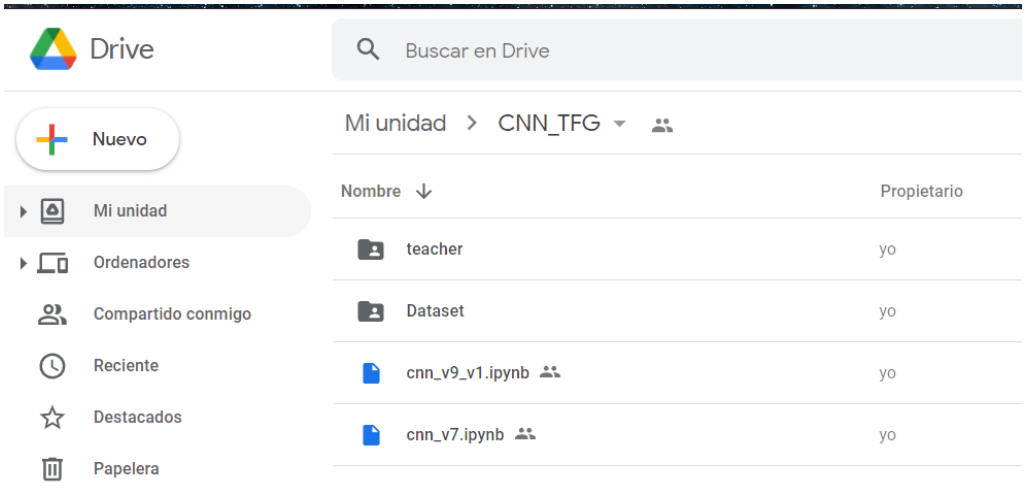


Figura 13 Repositorio Drive utilizado en el proyecto

- Visual Studio Code: Ha sido una de las primeras opciones para poder programar la interfaz, pero después se ha optado por trasladarse a un IDE especifico de Python por las ventajas que ofrece este frente a un IDE general como es Visual Studio Code.

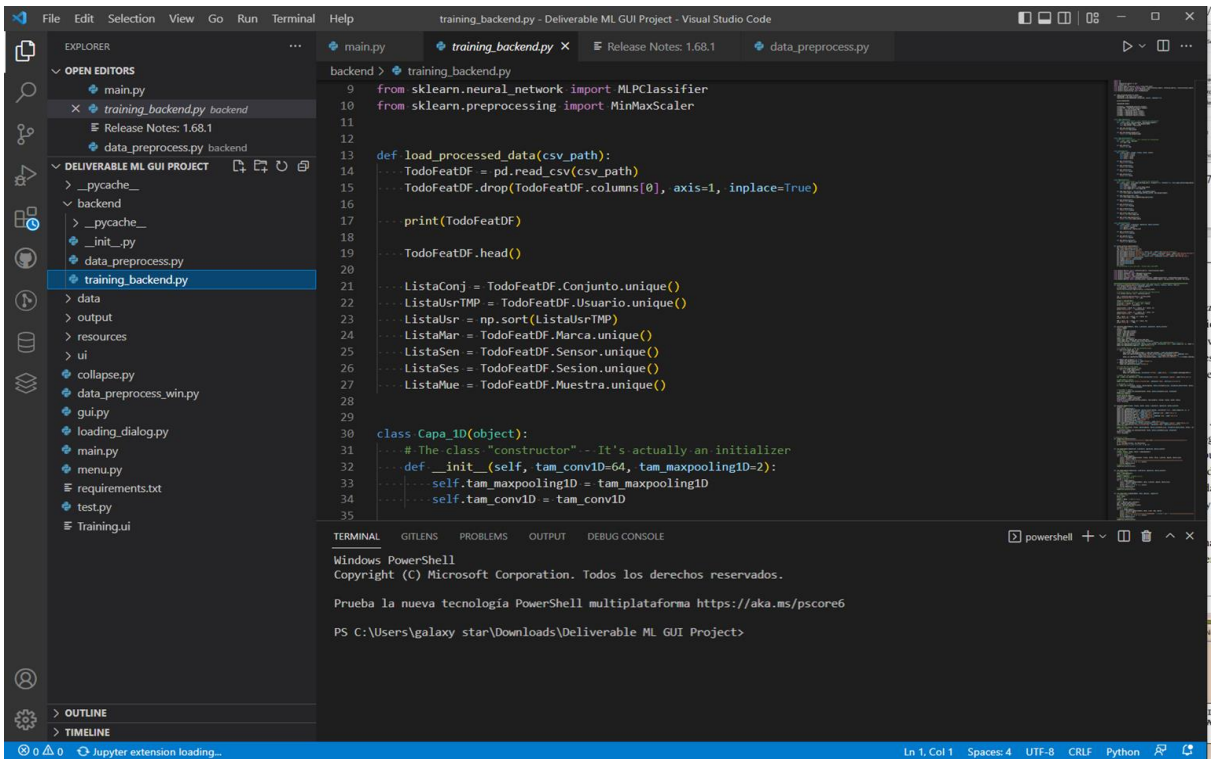


Figura 14 Editor de código Visual Studio Code

- *Pycharm*: Se ha elegido como IDE para desarrollar la interfaz de este proyecto, no solo es un editor de código, sino que también tiene depurador e interprete. Pycharm está especializado en Python tal como eclipse está especializado en Java.

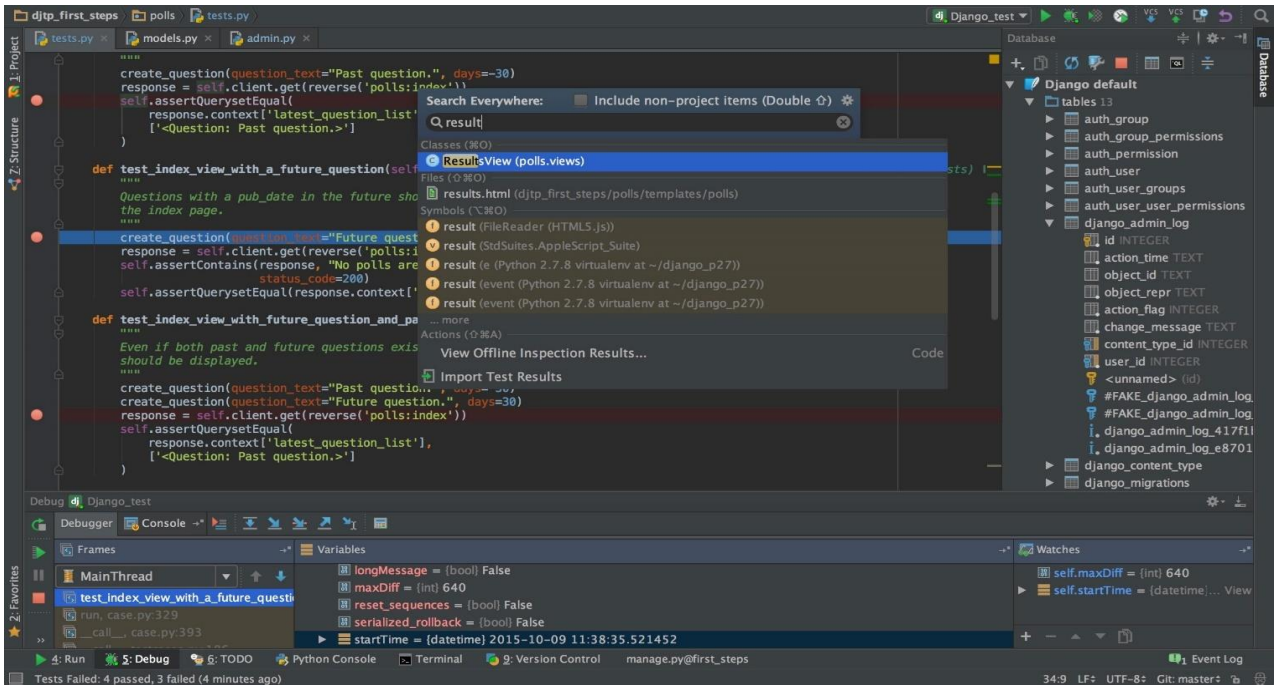


Figura 15 IDE pytorch

- *Microsoft Word:* Se ha elegido como editor de texto para realizar la documentación y memoria del proyecto

- *Microsoft Project:* Se ha utilizado para realizar la planificación del proyecto junto con el cálculo de costes y diagramas de Gantt

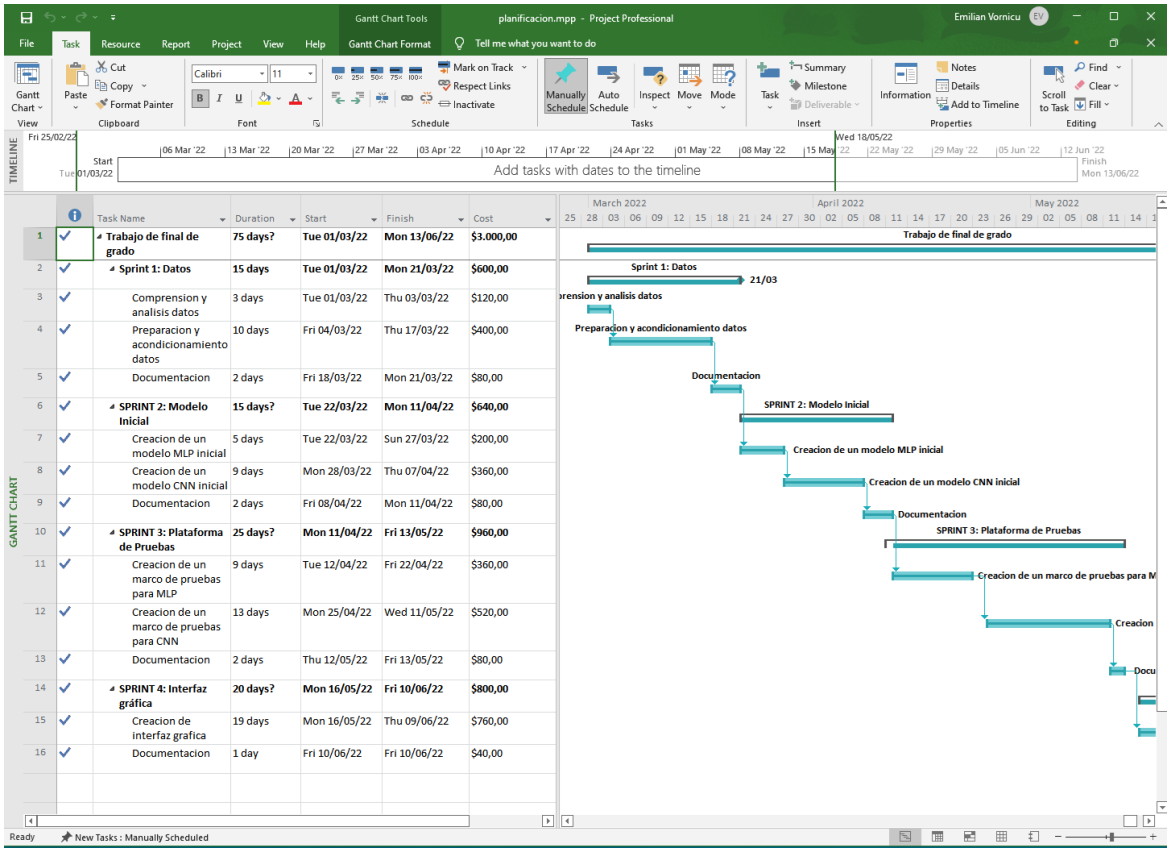


Figura 16 Plataforma MS Project

3.3. Sistema Operativo

En cuanto al sistema operativo, el ordenador utilizado para el desarrollo de este proyecto, dispone de tanto de Windows como de Linux. Destacamos que el proyecto puede realizarse en cualquiera de los sistemas operativos mencionados con anterioridad.

Para la realización de las diferentes fases del proyecto hemos utilizado ambos sistemas operativos:

- Windows: Hemos utilizado Windows para la realización y el desarrollo tanto de código como de documentación de los tres primero Sprints, detallados en más profundidad en la sección 4.2.

- Linux: Para la creación de la interfaz hemos preferido utilizar Linux con el IDE pycharm descrito con anterioridad

3.4. Lenguaje de programación

Para el desarrollo de este proyecto se ha optado por Python, tanto para el desarrollo del *back*end como para el *frontend* o desarrollo de la interfaz.

Python es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. A diferencia de otros lenguajes como Java o .NET, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en Python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado interpretador, por lo que no es necesario "traducirlo" a lenguaje máquina. Es un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional.

El motivo principal para optar por Python como lenguaje de programación para este proyecto es la multitud de bibliotecas disponibles que facilitan el código de algoritmos de Aprendizaje Automático.

3.5. Bibliotecas

Uno de los motivos para la elección de Python como lenguaje de programación principal para este proyecto es el soporte que tiene de librerías para la implementación de Redes Neuronales y Aprendizaje Automático. De entre estas librerías destacamos TensorFlow, Scikit-learn, Keras… también se recibe soporte para el tratamiento y almacenamiento de datos, destacamos Pandas, Numpy etc . Para crear gráficos hemos utilizado matplotlib y para llamar al sistema hemos utilizado OS.

En la parte del desarrollo de la aplicación, hemos usado PyQt5 y sus diferentes módulos. PyQt5 se define como un binding de la biblioteca gráfica QT para el lenguaje de programación Python. Y permite crear interfaces gráficas con Python.

A continuación, se muestran algunas de las librerías más importantes utilizadas en el proyecto y su versión:

- Numpy (versión 1.22.4): es una biblioteca para el lenguaje de programación Python que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas facilita el uso de comandos desde los cuadernos o scripts de Python.

- Pandas (versión 1.4.2): pandas es una biblioteca de software escrita como extensión de Numpy para manipulación y análisis de datos para el lenguaje de programación Python. En particular, ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.

- OS: El módulo os de Python nos permite realizar operaciones dependientes del Sistema Operativo como crear una carpeta, listar contenidos de una carpeta, conocer acerca de un proceso, finalizar un proceso, etc

- Re: Modulo para las expresiones regulares. Estas se usan normalmente en aplicaciones que implican procesamiento de una gran cantidad de texto.

- Matplotlib (versión 3.5.2): Matplotlib es una librería de Python especializada en la creación de gráficos en dos dimensiones.

- scikit-learn (versión 1.1.1): es una librería de Aprendizaje Automático para

el lenguaje de programación Python, que incluye varios algoritmos de clasificación, regresión y análisis de grupos.

- TensorFlow (versión 2.9.1): es una plataforma de código abierto utilizada para Machine Learning y para Aprendizaje Profundo.

- Keras (versión 2.9.0): está diseñada para experimentar con Redes Neuronales. Puede ejecutarse sobre TensorFlow.

- PyQt5 (versión 5.15.7): es una biblioteca gráfica para el lenguaje de programación Python que permite crear interfaces gráficas.

Todas las librerías utilizadas en el proyecto y su respectiva versión se muestran en el Anexo X

Capítulo 4
Metodología

La metodología de gestión de proyectos es la disciplina del conocimiento que se encarga de elaborar, definir y sistematizar el conjunto de técnicas, métodos y procedimientos que se deben seguir durante el desarrollo de un proyecto [17].

4.1. Metodologías de proyecto

En la actualidad se pueden diferenciar dos grandes grupos de metodologías de desarrollo de software: las ágiles y las tradicionales.

4.1.1. Metodologías tradicionales

Las metodologías de desarrollo de software tradicionales se caracterizan por definir total y rígidamente los requisitos al inicio de los proyectos de ingeniería de software. La organización del trabajo de las metodologías tradicionales es lineal, es decir, las etapas se suceden una tras otra y no se puede empezar la siguiente sin terminar la anterior.

Entre las metodologías de desarrollo software tradicionales más conocidas son:

- *Waterfall (cascada):* es una metodología en la que las etapas se organizan de arriba abajo. Se desarrollan las diferentes funciones en etapas diferenciadas y obedeciendo un riguroso orden. Antes de cada etapa se debe revisar el producto para ver si está listo para pasar a la siguiente fase. Los requisitos y especificaciones iniciales no están predispuestos para cambiarse, por lo que no se pueden ver los resultados hasta que el proyecto ya esté bastante avanzado.

- *Prototipado*: se basa en la construcción de un prototipo de software que se construye rápidamente para que los usuarios puedan probarlo y aportar feedback. Así, se puede arreglar lo que está mal e incluir otros requerimientos que puedan surgir. Es un modelo iterativo que se basa en el método de prueba y error para comprender las especificidades del producto.

- Espiral: es una combinación de los dos modelos anteriores, que añade el concepto de análisis de riesgo. Se divide en cuatro etapas: planificación, análisis de riesgo, desarrollo de prototipo y evaluación del cliente. El nombre de esta metodología da nombre a su funcionamiento, ya que se van procesando las etapas en forma de espiral. Cuanto más cerca del centro se está, más avanzado está el proyecto.

- *Incremental*: en esta metodología de desarrollo de software se va construyendo el producto final de manera progresiva. En cada etapa incremental se agrega una nueva funcionalidad, lo que permite ver resultados de una forma más rápida en comparación con el modelo en cascada. El software se puede empezar a utilizar incluso antes de que se complete totalmente y, en general, es mucho más flexible que las demás metodologías.

- *Diseño rápido de aplicaciones* (RAD): esta metodología permite desarrollar software de alta calidad en un corto periodo de tiempo. Los costes son mucho más altos y el desarrollo más flexible, aunque requiere una mayor intervención de los usuarios. Por otro lado, el código puede contener más errores, y sus funciones son limitadas debido al poco tiempo del que se dispone para desarrollarlas. El objetivo es iterar el menor número posible de veces para conseguir una aplicación completa de forma rápida.

4.1.2. Metodologías ágiles

Las metodologías ágiles de desarrollo de software son las más utilizadas hoy en día debido a su alta flexibilidad y agilidad. Los equipos de trabajo que las utilizan son mucho más productivos y eficientes, ya que saben lo que tienen que hacer en cada momento. Además, la metodología permite adaptar el software a las necesidades que van surgiendo por el camino, lo que facilita construir aplicaciones más funcionales.

- Kanban: Consiste en dividir las tareas en porciones mínimas y organizarlas en un tablero de trabajo dividido en tareas pendientes, en curso y finalizadas. De esta forma, se crea un flujo de trabajo muy visual basado en tareas prioritarias e incrementando el valor del producto.

- Scrum: es también una metodología incremental que divide los requisitos y tareas de forma similar a Kanban. Se itera sobre bloques de tiempos cortos y fijos (entre dos y cuatro semanas) para conseguir un resultado completo en cada iteración. Las etapas son: planificación de la iteración (planning sprint), ejecución (sprint), reunión diaria (daily meeting) y demostración de resultados (sprint review). Cada iteración por estas etapas se denomina también sprint.

- Lean: está configurado para que pequeños equipos de desarrollo muy capacitados elaboren cualquier tarea en poco tiempo. Los activos más importantes son las personas y su compromiso, relegando así a un segundo plano el tiempo y los costes. El aprendizaje, las reacciones rápidas y potenciar el equipo son fundamentales.

- Programación extrema (XP): es una metodología de desarrollo de software basada en las relaciones interpersonales, que se consideran la clave del éxito. Su principal objetivo es crear un buen ambiente de trabajo en equipo y que haya un feedback constante del cliente. El trabajo se basa en 12 conceptos: diseño sencillo, testing, refactorización y codificación con estándares, propiedad colectiva del código, programación en parejas, integración continua, entregas semanales e integridad con el cliente, cliente in situ, entregas frecuentes y planificación.

4.2. Metodología utilizada

Debido a la complejidad y características del proyecto, para la realización del Trabajo de fin de grado, se ha seguido la metodología *SCRUM* perteneciente a la metodología ágil, mencionada con anterioridad. Por lo que en primer lugar hemos realizado un *Sprit Planning* o planificación de sprint en los que se ha fijado el

objetivo del proyecto y hemos analizado el alcance de este. Regularmente se han mantenido *dailys* o reuniones con el profesor, comentando los posibles bloqueos y avances y por último hemos realizado *Sprint Reviews* después de cada Sprint, valorando los cambios realizados.

Tal como se ha comentado, se ha divido el proyecto en cuatro *sprints*, con las siguientes épicas que se detallan a continuación:

- Sprint 1: Datos.
    - Épica 1: Comprensión y análisis de la naturaleza de los datos
    - Épica 2: Preparación y acondicionamiento de los datos
    - Épica 3: Documentación
- Sprint 2: Modelo Inicial
    - Épica 1: Creación de un modelo MLP inicial
    - Épica 2: Creación de un modelo CNN inicial
    - Épica 3: Documentación
- Sprint 3: Plataforma de Pruebas
    - Épica 1: Creación de un marco de pruebas para MLP
    - Épica 2: Creación de un marco de pruebas para CNN
    - Épica 3: Documentación
- Sprint 4: Interfaz gráfica
    - Épica 1: Creación de interfaz gráfica
    - Épica 2: Documentación

Cada Sprint de este proyecto está organizado de la siguiente manera:

- Objetivos del Sprint: Se mencionan los objetivos del Sprint
- Análisis de los objetivos del Sprint: Se analiza los objetivos a cumplir para llegar a la solución
- Desarrollo e implementación de la solución: Se muestra la implementación y desarrollo de las tareas correspondientes al sprint
- Pruebas de la solución implementada: Se verifica que la solución implementada es efectiva mostrando si la tarea lo permite las ejecuciones de estas o los entregables.

Capítulo 5

Planificación

El presente TFG constituye 12 European Credit Transder System (ECTS) de la formación académica de la titulación de Grado en Ingeniería Informática [9]. Teniendo en cuenta el actual marco de traducción a horas de la UVA, 1 ECTS supone la inversión de 25 horas de trabajo por tanto esto implicaría una dedicación estimada de 300 horas.

El inicio data del 1 de marzo de 2022 y su fecha estimada de entrega es el 13 de junio. Por lo que, tendríamos un marco temporal de 75 días con una carga promedio de trabajo de 20 horas semanales.

A continuación, se presenta la planificación inicial del proyecto, las variaciones que sufrió y el coste del proyecto

5.1. Planificación inicial

Una de las primeras tareas en el desarrollo de este trabajo fue realizar la planificación del proyecto. Para ello se desglosaron en la sección 4.2 las distintas épocas y taras de este proyecto. La planificación inicial del proyecto fue la siguiente:

Task Name	Duration	Start	Finish
Inicio del Proyecto	**75 days?**	**Tue 01/03/22**	**Mon 13/06/22**
Sprint 1: Datos	**15 days**	**Tue 01/03/22**	**Mon 21/03/22**
Comprension y analisis datos	3 days	Tue 01/03/22	Thu 03/03/22
Preparacion y acondicionamiento datos	10 days	Fri 04/03/22	Thu 17/03/22
Documentacion	2 days	Fri 18/03/22	Mon 21/03/22
SPRINT 2: Modelo Inicial	**15 days?**	**Tue 22/03/22**	**Mon 11/04/22**
Creacion de un modelo MLP inicial	5 days	Tue 22/03/22	Sun 27/03/22
Creacion de un modelo CNN inicial	9 days	Mon 28/03/22	Thu 07/04/22
Documentacion	2 days	Fri 08/04/22	Mon 11/04/22
SPRINT 3: Plataforma de Pruebas	**25 days?**	**Mon 11/04/22**	**Fri 13/05/22**
Creacion de un marco de pruebas para MLP	9 days	Tue 12/04/22	Fri 22/04/22
Creacion de un marco de pruebas para CNN	13 days	Mon 25/04/22	Wed 11/05/22
Documentacion	2 days	Thu 12/05/22	Fri 13/05/22
SPRINT 4: Interfaz gráfica	**20 days?**	**Mon 16/05/22**	**Fri 10/06/22**
Creacion de interfaz grafica	19 days	Mon 16/05/22	Thu 09/06/22
Documentacion	1 day	Fri 10/06/22	Fri 10/06/22

Tabla 1 Planificacion inicial del proyecto

Tal como se puede observar en la figura XX, en la vista de actividades, el proyecto da Inicio el 1 de marzo de 2022 y se preveía que acabara el 13 de junio de 2022, con una duración total de 75 días. Para poder llegar al objetivo de 300 h mencionado en la introducción de este capítulo se ha establecido un trabajo diario de 4h o 20h semanales respectivamente. La documentación de la memoria se ha realizado de manera gradual al final de cada Sprint.

El Sprint 1: Datos, su objetivo es la preparación y procesado de los datos ponibles da comienzo el 1 de marzo y acaba el 21 del mismo mes, tiene una duración de 15 días. Y se compone de 3 tareas tal como se ha indicado en la sección 4.2. La tarea más importante de este sprint es la Preparación y acondicionamiento datos que ocupa 10 días o 40 h respectivamente. A continuación, se desglosa la planificación de este Sprint:

Sprint 1: Datos	**15 days**	**Tue 01/03/22**	**Mon 21/03/22**
Comprension y analisis datos	3 days	Tue 01/03/22	Thu 03/03/22
Preparacion y acondicionamiento datos	10 days	Fri 04/03/22	Thu 17/03/22
Documentacion	2 days	Fri 18/03/22	Mon 21/03/22

Tabla 2 Planificación inicial del Sprint 1

El Sprint 2: Modelo inicial, consiste en la creación de los modelos estándar, que nos servirán de base para la plataforma de pruebas. Da comienzo el 22 de marzo y acaba el 11 de abril, tiene una duración de 15 días. Y se compone de 3 tareas tal como se ha indicado en la sección 4.2. La tarea más importante de este sprint es la creación de Preparación y acondicionamiento datos que ocupa 9 días o 36 h respectivamente.
A continuación, se desglosa la planificación de este Sprint:

SPRINT 2: Modelo Inicial	**15 days?**	**Tue 22/03/22**	**Mon 11/04/22**
Creacion de un modelo MLP inicial	5 days	Tue 22/03/22	Sun 27/03/22
Creacion de un modelo CNN inicial	9 days	Mon 28/03/22	Thu 07/04/22
Documentacion	2 days	Fri 08/04/22	Mon 11/04/22

Tabla 3 Planificación inicial del Sprint 2

El Sprint 3 Plataforma de Pruebas, consta en modificar los modelos previamente creados para permitir al usuario seleccionar los parámetros de los modelos MLP y CNN. Este sprint da comienzo el 11 de abril y acaba el 15 de mayo, tiene una duración de 25 días. Este sprint es uno de los más delicados porque en él se basa la funcionalidad de la aplicación. Además, hay que elegir con cuidado los parámetros que puede elegir el usuario porque las redes

CNN son muy sensibles. El sprint se compone de 3 tareas. La tarea más importante de este sprint es la creación del marco de pruebas de las CNN debido a la cantidad de parámetros que tienen frente a las MLP.
A continuación, se desglosa la planificación de este Sprint:

SPRINT 3: Plataforma de Pruebas	**25 days?**	**Mon 11/04/22**	**Fri 13/05/22**
Creacion de un marco de pruebas para MLP	9 days	Tue 12/04/22	Fri 22/04/22
Creacion de un marco de pruebas para CNN	13 days	Mon 25/04/22	Wed 11/05/22
Documentacion	2 days	Thu 12/05/22	Fri 13/05/22

Tabla 4 Planificación inicial del Sprint 3

El Sprint 4 Interfaz gráfica, consta de crear una interfaz interactiva para que el usuario pueda preprocesar el modelo, y entrenarlo a su antojo. Este sprint junto con el sprint anterior son los que más tiempo llevan, un factor añadido es que no he programado interfaces en Python. Este sprint se compone de 2 tareas. Este sprint da comienzo el 16 de mayo y acaba el 10 de junio, tiene una duración de 20 días. A continuación, se desglosa la planificación de este Sprint:

SPRINT 4: Interfaz gráfica	**20 days?**	**Mon 16/05/22**	**Fri 10/06/22**
Creacion de interfaz grafica	19 days	Mon 16/05/22	Thu 09/06/22
Documentacion	1 day	Fri 10/06/22	Fri 10/06/22

Tabla 5 Planificación inicial del Sprint 4

El **diagrama de Gantt** es una herramienta gráfica cuyo objetivo es exponer el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado. Este es de gran ayuda en la planificación de proyectos software. A continuación, en la figura XX se muestra el diagrama de Gantt que sigue

la planificación expuesta con anterioridad.

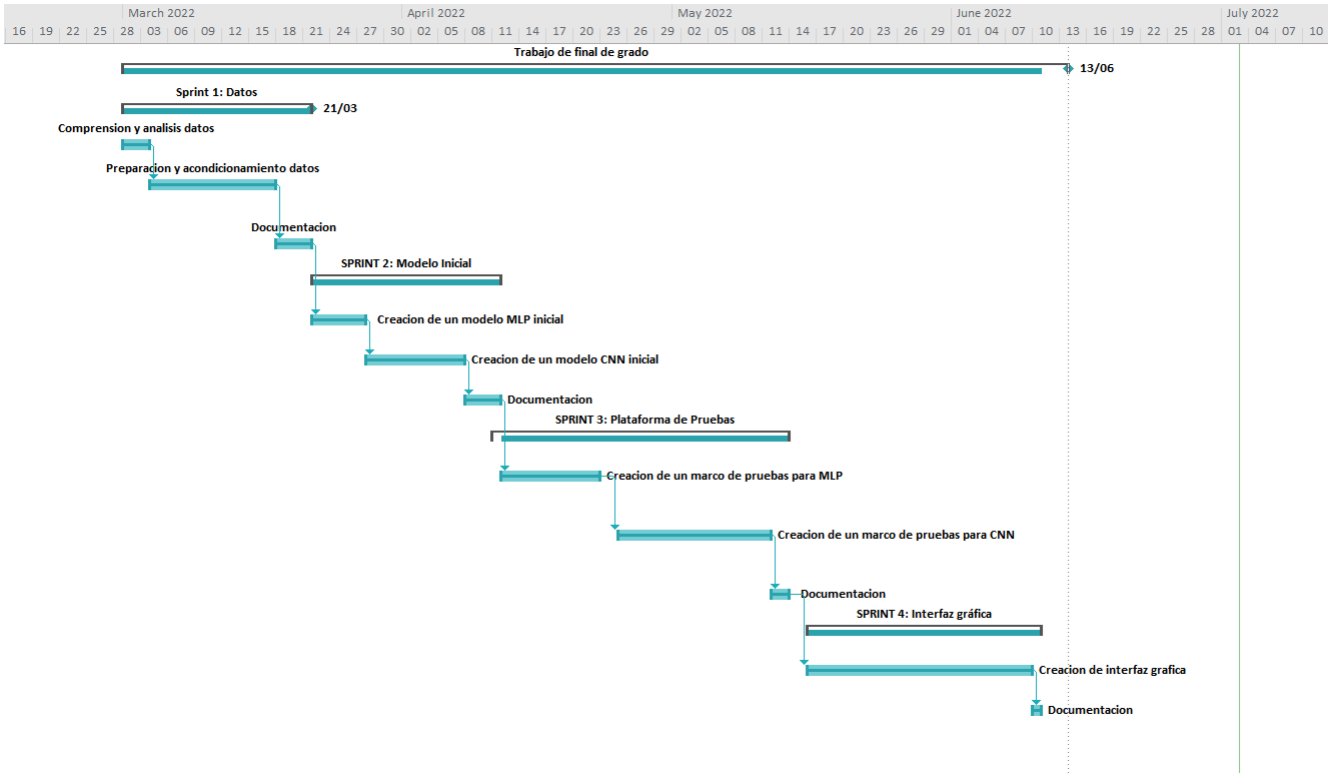


Figura 17 Diagrama de Gannt del proyecto

5.2. Estimación del coste del proyecto

5.2.1. Coste software

De acuerdo con la organización del Sprint mostrada en la *tabla 6* se ha realizado un cálculo del coste del proyecto.

Task Name	Duration	Start	Finish	Cost
Inicio del Proyecto	**75 days?**	**Tue 01/03/22**	**Mon 13/06/22**	**$3.000,00**
Sprint 1: Datos	**15 days**	**Tue 01/03/22**	**Mon 21/03/22**	**$600,00**
Comprension y analisis datos	3 days	Tue 01/03/22	Thu 03/03/22	$120,00
Preparacion y acondicionamiento datos	10 days	Fri 04/03/22	Thu 17/03/22	$400,00
Documentacion	2 days	Fri 18/03/22	Mon 21/03/22	$80,00
SPRINT 2: Modelo Inicial	**15 days?**	**Tue 22/03/22**	**Mon 11/04/22**	**$640,00**
Creacion de un modelo CNN inicial	5 days	Tue 22/03/22	Sun 27/03/22	$200,00
Creacion de un modelo MLP inicial	9 days	Mon 28/03/22	Thu 07/04/22	$360,00
Documentacion	2 days	Fri 08/04/22	Mon 11/04/22	$80,00
SPRINT 3: Plataforma de Pruebas	**25 days?**	**Mon 11/04/22**	**Fri 13/05/22**	**$960,00**
Creacion de un marco de pruebas para MLP	9 days	Tue 12/04/22	Fri 22/04/22	$360,00
Creacion de un mardo de pruebas para CNN	13 days	Mon 25/04/22	Wed 11/05/22	$520,00
Documentacion	2 days	Thu 12/05/22	Fri 13/05/22	$80,00
SPRINT 4: Interfaz gráfica	**20 days?**	**Mon 16/05/22**	**Fri 10/06/22**	**$800,00**
Creacion de interfaz grafica	19 days	Mon 16/05/22	Thu 09/06/22	$760,00
Documentacion	1 day	Fri 10/06/22	Fri 10/06/22	$40,00

Tabla 6 Planificación de coste de software del proyecto

Para la realización del cálculo del coste del proyecto, se han tenido en cuenta varios factores:

- El coste laboral. El precio se ha establecido a 10€ la hora que sería aproximadamente 20000€ al año. El precio medio de un ingeniero informático en España es de 29000 o 14,97 por hora.
- Duración del día laborable. Para llegar al objetivo de 300h mencionado en la introducción de este capítulo se ha establecido que cada día laborable sea de 4 h. Siendo la duración total del proyecto de 75 días.

5.3. Gestión de riesgos

Al igual que en cualquier otro proceso organizacional, en los proyectos de software es necesario realizar una adecuada gestión de riesgos. El objetivo de este apartado es identificar, abordar y mitigar elementos de riesgo antes de que se conviertan en una amenaza para la ejecución exitosa del proyecto y para el logro de los objetivos planteados.

A continuación, se muestran los posibles riesgos que se pueden plantear durante la ejecución de este proyecto con su probabilidad e impacto:

Id	Categoría	Descripción	Probabilidad	Impacto
R001	Requisitos	Alcance u objetivos poco claros	baja	medio
R004	Requisitos	Las expectativas de los usuarios finales son interpretadas erróneamente	baja	medio
R002	Desarrollo	Estimación inadecuada del tiempo de ejecución de las épicas	media	medio
R003	Equipo	Contagio de COVID-19 de alguno de los miembros del equipo.	baja	bajo
R004	Desarrollo	Dificultad en la comprensión del problema	baja	bajo
R005	Desarrollo	Tiempos de ejecución elevados	baja	bajo
R005	Equipo	Falta de tiempo para el desarrollo del proyecto	baja	medio

Tabla 7 Gestión de Riesgos del proyecto

5.4. Variaciones respecto a la planificación inicial

En la planificación inicial del proyecto se preveía que la fecha de inicio de este sería el 1 de marzo de 2022 y la fecha estimada de entrega sería el 13 de junio. Esta estimación calculaba que el proyecto se realizaría a lo largo de 75 días con una carga de trabajo de 4 h por día.

Desafortunadamente durante el proyecto han surgido imprevistos y se han materializado algunos de los riesgos mencionados en la sección anterior, retrasando así la fecha de inicio, el desarrollo de este y la fecha de entrega de este proyecto.

Uno de los riesgos que ha contribuido considerablemente al retraso en el desarrollo e inicio de las épicas ha sido el riesgo *R004 Dificultad en la comprensión del problema* que va estrechamente ligado con el riesgo *R002 Estimación inadecuada del tiempo de ejecución de las épicas*.

Durante el primer sprint, que es el encargado del procesamiento de los datos, se ha dedicado mayor tiempo del esperado en la comprensión de la naturaleza de los datos y en cómo realizar el procesamiento de los datos en crudo. Lo mismo sucedió en el sprint 2 con la creación del modelo CNN debido a que no se había trabajado con este modelo anteriormente y es muy sensible a los parámetros que se seleccionan.

Algo semejante ha sucedido en el último sprint que tiene como objetivo la creación de un interfaz basado el marco de pruebas creado previamente, este sprint ha durado más de lo esperado porque no se ha trabajado previamente con Python para hacer interfaces, por lo que se tendría que haber tenido en cuenta esto a la hora de estimar la actividad.

Otro riesgo materializado ha sido el *R005 Falta de tiempo para el desarrollo del proyecto,* aunque el desarrollador del proyecto contaba con los días indicados para poder desarrollar el proyecto, en varias ocasiones al trabajar de jornada completa, surgían imprevistos laborales por lo que las sesiones se extendían reduciendo el tiempo planificado para el de desarrollo del proyecto. Estas circunstancias han extendido la duración del proyecto y lo han hecho más complicado desde un punto de vista no solo temporal sino también desde un punto de vista mental y emocional.

Capítulo 6
Datos

6.1. Descripción de los datos

Los datos de este proyecto pertenecen a usuarios de dispositivos ponibles. Los dispositivos ponibles son aquellos dispositivos que "se llevan puestos" y reciben este nombre de la palabra inglesa wearable. Estos dispositivos son responsables de registrar datos acerca de la actividad física realizada por las personas.

La estructura de los datos recogidos se separa en dos grandes grupos de usuarios diferentes. De un lado tenemos los usuarios del grupo TFG y por otro lado tenemos a los usuarios del conjunto TFM. La estructura mencionada es la siguiente:

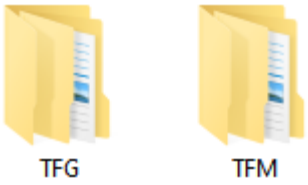


Figura 18 Estructura de los conjuntos de datos ponibles

Dentro de cada conjunto, se encuentran los diferentes usuarios. El nombre que reciben estos no necesariamente coincide con el número de usuarios totales, y no significa que falten usuarios, es mera cuestión de nomenclatura como sucede en conjunto de usuarios TFG.

El conjunto TFG dispone de datos de 12 usuarios distintos, tal como se muestra a continuación.

Nombre	Fecha de modificación	Tipo	Tamaño
usuario1	13/02/2022 16:31	Carpeta de archivos	
usuario2	13/02/2022 16:31	Carpeta de archivos	
usuario3	13/02/2022 16:31	Carpeta de archivos	
usuario8	13/02/2022 16:31	Carpeta de archivos	
usuario9	13/02/2022 16:31	Carpeta de archivos	
usuario10	13/02/2022 16:31	Carpeta de archivos	
usuario11	13/02/2022 16:31	Carpeta de archivos	
usuario12	13/02/2022 16:31	Carpeta de archivos	
usuario13	13/02/2022 16:31	Carpeta de archivos	
usuario14	13/02/2022 16:31	Carpeta de archivos	
usuario15	13/02/2022 16:31	Carpeta de archivos	
usuario18	13/02/2022 16:31	Carpeta de archivos	

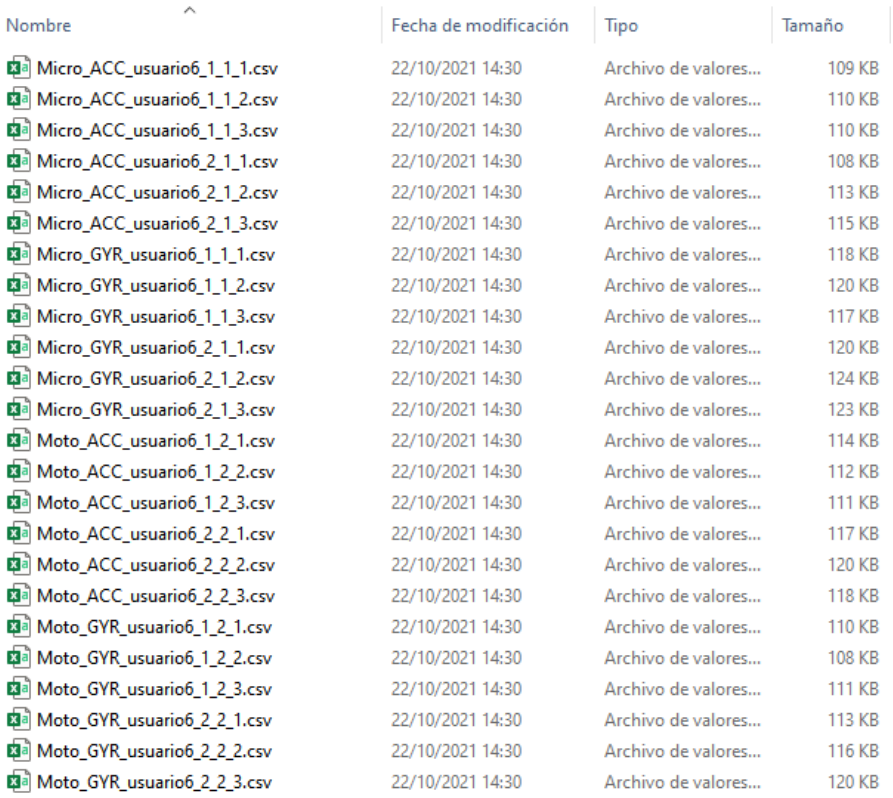
Figura 19 Estructura de directorios del conjunto TFG

El conjunto TFM dispone de datos de 24 usuarios distintos, como se muestra a continuación:

Nombre	Fecha de modificación	Tipo	Tan
usuario1	13/02/2022 16:31	Carpeta de archivos	
usuario2	13/02/2022 16:31	Carpeta de archivos	
usuario3	13/02/2022 16:31	Carpeta de archivos	
usuario4	13/02/2022 16:31	Carpeta de archivos	
usuario5	13/02/2022 16:31	Carpeta de archivos	
usuario6	13/02/2022 16:31	Carpeta de archivos	
usuario7	13/02/2022 16:31	Carpeta de archivos	
usuario8	13/02/2022 16:31	Carpeta de archivos	
usuario9	13/02/2022 16:31	Carpeta de archivos	
usuario10	13/02/2022 16:31	Carpeta de archivos	
usuario11	13/02/2022 16:31	Carpeta de archivos	
usuario12	13/02/2022 16:31	Carpeta de archivos	
usuario13	13/02/2022 16:31	Carpeta de archivos	
usuario14	13/02/2022 16:31	Carpeta de archivos	
usuario15	13/02/2022 16:31	Carpeta de archivos	
usuario16	13/02/2022 16:31	Carpeta de archivos	
usuario17	13/02/2022 16:31	Carpeta de archivos	
usuario18	13/02/2022 16:31	Carpeta de archivos	
usuario19	13/02/2022 16:31	Carpeta de archivos	
usuario20	13/02/2022 16:31	Carpeta de archivos	
usuario21	13/02/2022 16:31	Carpeta de archivos	
usuario22	13/02/2022 16:31	Carpeta de archivos	
usuario23	13/02/2022 16:31	Carpeta de archivos	
usuario24	13/02/2022 16:31	Carpeta de archivos	

Figura 20 Estructura de directorios del conjunto TFM

En cada carpeta de cada usuario se encuentran los datos recogidos de cada dispositivo ponible. Por ejemplo, estos son los datos recogidos del usuario 6 del conjunto de datos TFM:



Nombre	Fecha de modificación	Tipo	Tamaño
Micro_ACC_usuario6_1_1_1.csv	22/10/2021 14:30	Archivo de valores...	109 KB
Micro_ACC_usuario6_1_1_2.csv	22/10/2021 14:30	Archivo de valores...	110 KB
Micro_ACC_usuario6_1_1_3.csv	22/10/2021 14:30	Archivo de valores...	110 KB
Micro_ACC_usuario6_2_1_1.csv	22/10/2021 14:30	Archivo de valores...	108 KB
Micro_ACC_usuario6_2_1_2.csv	22/10/2021 14:30	Archivo de valores...	113 KB
Micro_ACC_usuario6_2_1_3.csv	22/10/2021 14:30	Archivo de valores...	115 KB
Micro_GYR_usuario6_1_1_1.csv	22/10/2021 14:30	Archivo de valores...	118 KB
Micro_GYR_usuario6_1_1_2.csv	22/10/2021 14:30	Archivo de valores...	120 KB
Micro_GYR_usuario6_1_1_3.csv	22/10/2021 14:30	Archivo de valores...	117 KB
Micro_GYR_usuario6_2_1_1.csv	22/10/2021 14:30	Archivo de valores...	120 KB
Micro_GYR_usuario6_2_1_2.csv	22/10/2021 14:30	Archivo de valores...	124 KB
Micro_GYR_usuario6_2_1_3.csv	22/10/2021 14:30	Archivo de valores...	123 KB
Moto_ACC_usuario6_1_2_1.csv	22/10/2021 14:30	Archivo de valores...	114 KB
Moto_ACC_usuario6_1_2_2.csv	22/10/2021 14:30	Archivo de valores...	112 KB
Moto_ACC_usuario6_1_2_3.csv	22/10/2021 14:30	Archivo de valores...	111 KB
Moto_ACC_usuario6_2_2_1.csv	22/10/2021 14:30	Archivo de valores...	117 KB
Moto_ACC_usuario6_2_2_2.csv	22/10/2021 14:30	Archivo de valores...	120 KB
Moto_ACC_usuario6_2_2_3.csv	22/10/2021 14:30	Archivo de valores...	118 KB
Moto_GYR_usuario6_1_2_1.csv	22/10/2021 14:30	Archivo de valores...	110 KB
Moto_GYR_usuario6_1_2_2.csv	22/10/2021 14:30	Archivo de valores...	108 KB
Moto_GYR_usuario6_1_2_3.csv	22/10/2021 14:30	Archivo de valores...	111 KB
Moto_GYR_usuario6_2_2_1.csv	22/10/2021 14:30	Archivo de valores...	113 KB
Moto_GYR_usuario6_2_2_2.csv	22/10/2021 14:30	Archivo de valores...	116 KB
Moto_GYR_usuario6_2_2_3.csv	22/10/2021 14:30	Archivo de valores...	120 KB

Figura 21 Datos recogidos del usuario 6 del conjunto TFM

Cada usuario dispone de un numero diferente de datos, dependiendo de las sesiones y muestras que se han recogido con el dispositivo. Por tanto, la estructura de directorios general de los datos ponibles es la siguiente:

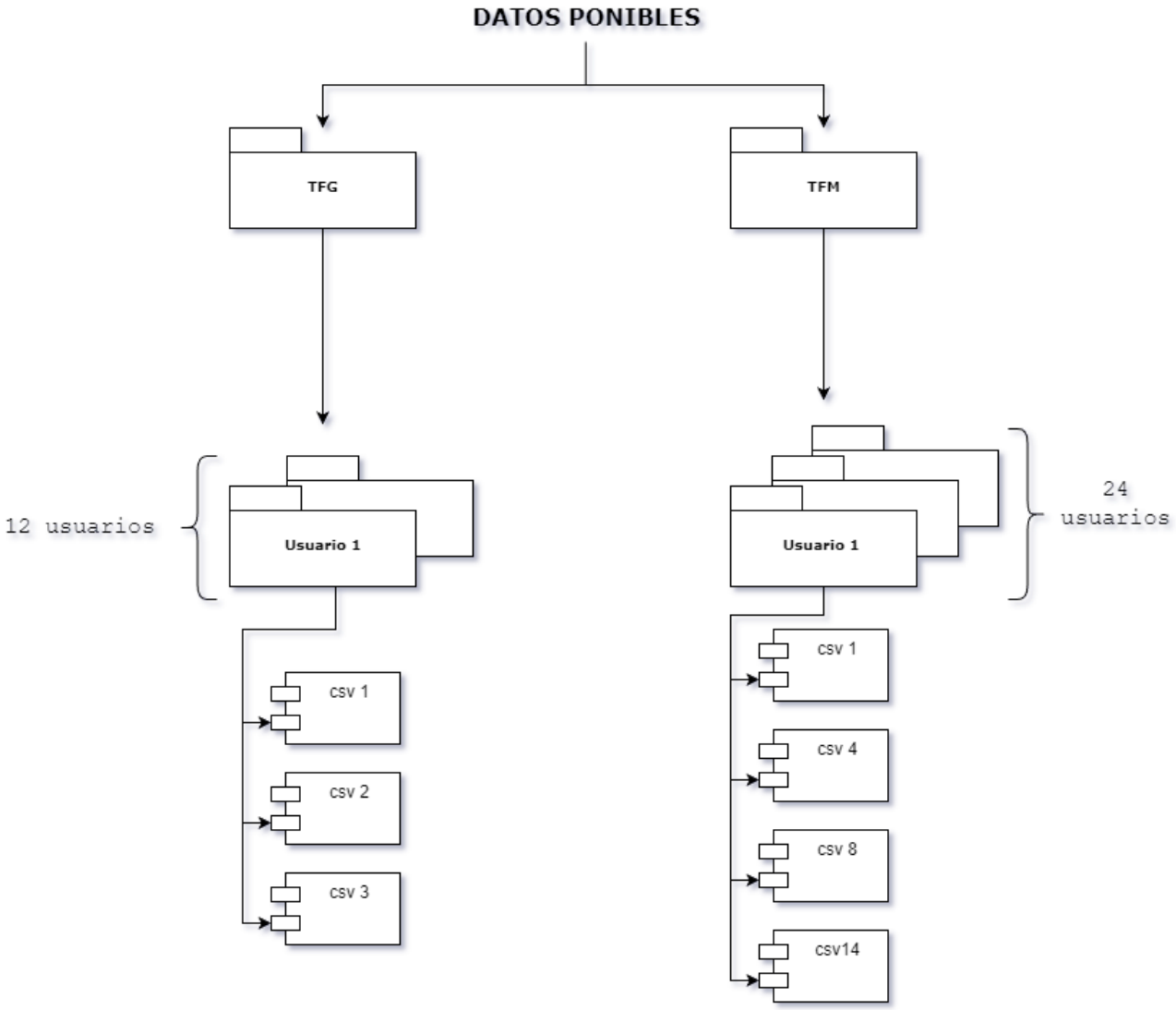


Figura 22 Estructura de datos ponibles

En la *figura 21* se han representado el árbol de directorios de los datos, cabe destacar que estos datos son datos en crudo y posteriormente se procesaran para poder ser utilizados en un modelo de inteligencia artificial.

Cada fichero csv mostrado tanto en la *figura 20* como en la *figura* 21 representan los datos captados por el dispositivo en una sesión determinada. Lo ficheros tienen la siguiente nomenclatura:

- Marca: Reprenda la marca del dispositivo ponible con el que se han capturado los datos, en este caso pueden ser Motorola o Microsoft, Moto o Micro respectivamente

- Sensor: Indica el sensor que se ha utilizado para captar los datos, puede ser Acelerómetro o Giróscopo, ACC o GYR respectivamente

- Usuario: Indica el nombre del usuario del que se han recogido los datos, debe coincidir con el nombre del directorio en el que se encuentra

- Sesión: Indica el número de sesión, puede ser 1 o 2
- Dispositivo: Indica el número dispositivo, puede ser 1 o 2
- Muestra: Indica el número de muestra, puede ser 1,2 o 3

Para ejemplificar lo ilustrado con anterioridad, en la siguiente *figura 22* se muestra el archivo de datos *Micro_ACC_usuario1_1_2_1* este csv se ha tomado con un dispositivo Microsoft, con el sensor acelerómetro, pertenece al usuario 1, es la sesión 1 del segundo dispositivo y es la primera muestra.

Figura 23 Ejemplo de nomenclatura de fichero de datos ponibles

En la siguiente sección se analizará el preprocesado que se realiza de los datos ponibles para que puedan ser utilizados como entrada de un modelo de IA.

6.2. Preprocesado

Tal como vimos en la sección anterior contamos con datos ponibles de dos conjuntos de usuarios diferentes, cada usuario tiene datos de diferentes marcas, sensores y muestras. Ahora analicemos que contiene cada fichero de datos. Por ejemplo, este es el contenido del fichero mostrado en la *figura 22*:

	A	B	C	D
	Timestamp	dato1	dato2	dato3
	30.0	319.939.026	51.25	234.085.373
	72.0	-2.195.122	-30.731.709	57.286.587
	124.0	94.390.244	-30.396.341	3.323.171
	122.0	59.603.661	-29.268.293	-62.804.878
	54.0	##########	-56.189.026	-97.347.565
	97.0	##########	-42.073.174	-50.731.709
	172.0	-70.304.878	-45.762.196	-49.695.122
	45.0	-22.530.489	-38.201.221	-27.621.952
	57.0	-71.554.878	-17.560.976	-15.030.488
	107.0	3.993.902	320.122	-16.585.365
	126.0	3.719.512	51.036.587	-0.396341
	83.0	92.195.122	64.756.096	2.487.805
	119.0	21.890.244	31.615.854	18.871.952
	70.0	11.676.829	-4.756.098	5.792.683
	84.0	-18.780.489	-29.207.317	-34.817.074
	106.0	-40.335.365	-14.908.537	-19.695.122
	130.0	-347.561	33.140.244	34.634.148

Figura 24 Contenido de un fichero en crudo de datos ponibles

Cada fichero se compone por un Timestamp o marca temporal y las tres coordenadas que se han captado con los sensores, eje X, Y, Z. El TimeStamp nos indica el intervalo de tiempo en milisegundos que ha pasado desde la última vez que se ha captado un dato. Por ejemplo, la segunda entrada tiene un TimeStamp de 72 ms e indica que desde la primera entrada hasta que se ha captado esta han pasado 72 milisegundos.

Para poder procesar la sesión previamente mostrada no nos vale con tener las coordenadas en instantes concretos de tiempo, necesitamos ventanas temporales para poder extraer información de los datos. Para reducir sesgos y debido a la naturaleza de los datos se ha decidido que los datos se procesaran en ventanas de 8 segundos.

Para evitar los sesgos que se pueden haber producido en la recogida de datos, establecemos también que el TimeStamp mínimo o tiempo que ha pasado entre la captura de dos datos no sea mayor de 200ms. Si el TimeStamp es menor que 200 ms asumimos que los datos son de fiar y los tenemos en cuenta, en caso contrario consideramos que ha habido un fallo al guardar el tiempo y reajustamos el TimeStamp como si hubiera habido un salto o diferencia de 100ms entre los datos.

Como deseamos que haya coherencia en el procesamiento de los datos las ventanas creadas y se eviten sesgos, estas tienen un solapamiento del 20% con respecto a ventanas anteriores.

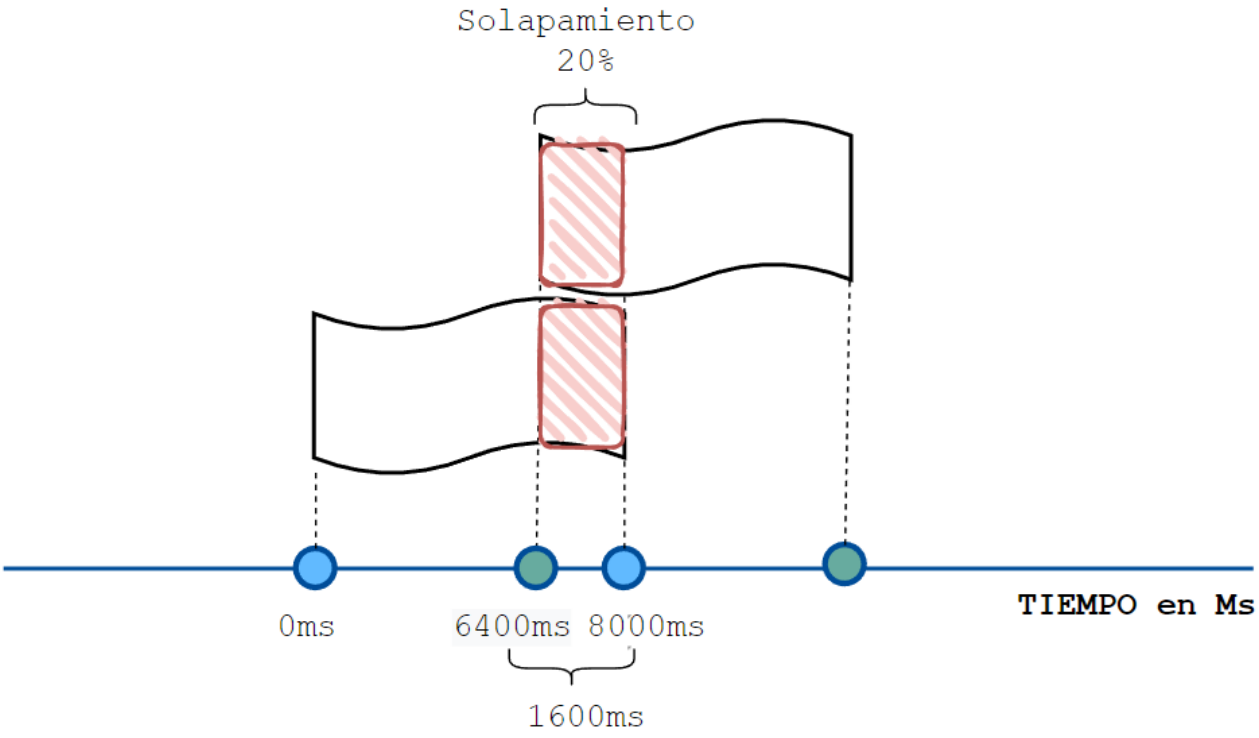


Figura 25 Ejemplo de solapamiento de ventanas

Tal como podemos observar en la *figura 24*, si deseamos que haya un 20% de solapamiento entre ventanas, la segunda ventana no empieza en 8000 ms, sino que ya tiene en cuenta las características de la ventana anterior empezando en 6400 ms. Por tanto, podemos decir que como cada ventana es de 8s y existe un solapamiento del 20% entre ventanas.

A cada ventana se le añaden las características que tiene cada fichero, entre las cuales tenemos:

- Numero de ventana: indica el número de ventana que se ha procesado
- Conjunto: indica el conjunto al que pertenece el fichero, TFG o TFM
- Usuario: Indica el nombre del usuario
- Marca: Indica la marca del dispositivo, Moto o Micro
- Sensor: Indica el sensor, ACC o GYR
- Sesión: Indica la sesión captada
- Muestra: Indica la muestra

Una vez procesado el fichero se pasa al siguiente fichero del conjunto. Tal como se muestra en la figura XX el objetivo es crear un Dataframe global que contenga los datos procesados de todos los usuarios del conjunto TFG y TFM.

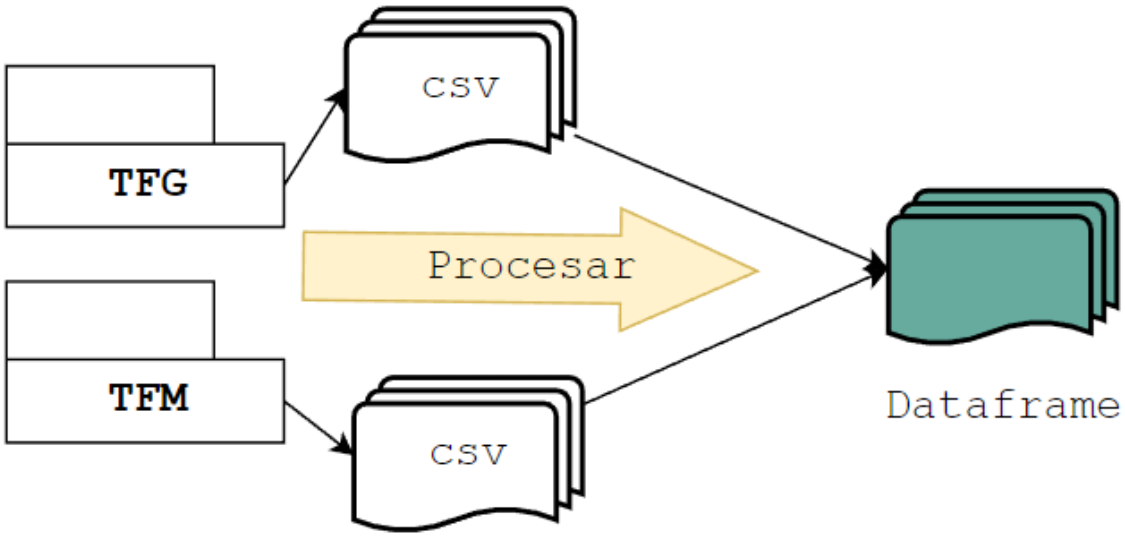


Figura 26 Descripcion del procesamiento de los conjuntos TFG y TFM

Este Dataframe se podrá guardar en formato csv y se podrá importar al sistema, por lo que el preprocesado de los datos que es una tarea que consume mucho tiempo y recursos solo se tendrá que realizar una sola vez.

6.3. División entrenamiento prueba

Una vez procesados los datos para que puedan ser usados en ambos modelos, tenemos que separarlos en dos conjuntos, el conjunto de entrenamiento y el conjunto de prueba

Para obtener dichos conjuntos se ha hecho lo siguiente. Suponiendo fijo el usuario, la marca (Microsoft o Motorola) y el sensor (ACC o GYR). Cada conjunto Entrenamiento o Prueba se ha dividido en dos grupos de usuarios, los usuarios auténticos o los impostores.

- En el conjunto de entrenamiento los usuarios auténticos son aquellos que pertenecen al conjunto de datos TFM, y tienen la muestra 1 de la sesión 1, y a estos se les asigna el valor 1. Entre los impostores del conjunto de entrenamiento se encuentran todos los usuarios del conjunto TFG, asignándoles un valor de 0.

- En el conjunto de prueba los usuarios auténticos son aquellos que pertenecen al conjunto de datos TFM, teniendo las muestras 1 y 2 de la sesión 2, y a estos se les asigna el valor 1. Entre los impostores del conjunto de entrenamiento se encuentran las muestras no usadas de del conjunto TFM, es decir aquellos que cuya sesión sea distinta de la muestra 1 de la sesión 1, asignándoles una salida con valor 0.

Capítulo 7
Construcción del sistema

Como se ha comentado en la sección 3.4 el lenguaje de programación elegido para este proyecto es Python, tanto para los modelos de aprendizaje profundo, como para la plataforma de pruebas e interfaz. Para la creación de los modelos de aprendizaje profundo utilizaremos los diferentes módulos y paquetes que ofrece Keras, que es un api construido sobre Tensor Flow que ofrece operaciones y modelos de alto nivel.

Como la metodología que hemos seguido para el desarrollo de este proyecto ha sido SCRUM, en este capítulo se analizan en detalle el desarrollo que se ha realizado en cada Sprint.

7.1. SPRINT 1: Datos

Como se ha explicado con detalle en el capítulo 6, los datos son una parte importante de nuestro proyecto, y necesitan un procesamiento previo para que puedan ser de utilidad. A continuación, explicaremos los objetivos de este sprint.

7.1.1. Objetivos

Los objetivos en Scrum, son metas tangibles redactadas en conjunto por el equipo que están sujetas a la duración del sprint [7]. Por tanto, el objetivo de este primer sprint es completar las 3 historias de usuario o épicas que se detallan a continuación:

- Sprint 1: Datos.
    - Épica 1: Comprensión y análisis de la naturaleza de los datos
    - Épica 2: Preparación y acondicionamiento de los datos
    - Épica 3: Documentación

7.1.2. Análisis de objetivos

Como se puede observar de la descripción anterior, en primer lugar, se analizan la naturaleza de los datos y el procesamiento de estos en un proyecto de Inteligencia Artificial. Para una mayor claridad esta *primera épica* se ha desollado en un capítulo aparte. El capítulo 6 analiza en detalle la estructura y naturaleza de los datos de este proyecto.

Como *segunda épica* tenemos la preparación y el acondicionamiento de los datos, en la que analizaremos el desarrollo e implementación de estos. Esta tarea es la más compleja de nuestro Sprint y se compone de varias subtareas que se detallan a continuación:

- Creación de una función capaz de procesar un fichero de datos ponibles
- Creación de función capaz de procesar los directorios TFG y TFM
- Creación de función capaz de guardar los datos ya procesados

- Creación de función capaz de importar los datos procesados
- Creación de función que separa los datos en entrenamiento y prueba

Por último, al igual que sucede en cada sprint tenemos como historia la documentación. Esta tarea consiste en redactar y documentar lo realizado a lo largo de este Sprint.

A continuación, se analiza cómo se ha realizado el desarrollo de la segunda épica de este sprint, *preparación y acondicionamiento de datos*.

7.1.3. Desarrollo e implementación

La primera subtarea, en la preparación y acondicionamiento de los datos, es la *Creación de una función capaz de procesar un fichero de datos ponibles.* Esta tarea, tal como su nombre indica, consiste en crear una función capaz de procesar los datos en crudos procedentes de los ficheros csv de cada usuario.

La operación de pre*procesar* es la encargada de realizar esta función. Esta función recibe como parámetro un dataframe inicializado, que es una variable global de nuestro programa y añade las características que nos permitirá crear una ventana que contendrá los datos de todos los usuarios al mismo tiempo.

Tal como explicamos en el capítulo 6, y como observamos en la figura XX los ficheros csv que se tienen que procesar están compuestos por coordenadas y marcas temporales o TimeStamps. Debido a la naturaleza temporal de los datos, para procesarlos no nos sirve tener las coordenadas en instantes concretos de tiempo, necesitamos ventanas temporales para poder extraer información de los datos. Tal como se explicó en el mismo capitulo para evitar sesgos entre las diferentes ventanas, también es necesario realizar un solapamiento de ventanas.

Timestamp	dato1	dato2	dato3
0.0	0.979004	-0.312256	0.174072
32.0	1.312.744	-0.525391	0.269775
103.0	1.400.391	-0.495605	-0.050049
122.0	1.045.166	-0.041992	0.38916

Tabla 8 Datos característicos de un fichero en crudo de ponibles

Nuestra función define en primer lugar el tamaño de las ventanas temporales y el solapamiento de estas, siendo estos valores de 8000 ms y de 1600 ms respectivamente. A continuación, para cada columna del fichero que se está procesando se realizan las siguientes acciones:

1. Se comprueba el TimeStamp mínimo o tiempo que ha pasado entre la captura de dos datos. Este no debe ser mayor de 200ms. Si el TimeStamp es menor que 200 ms el dato sigue con su proce-

samiento, en caso contrario consideramos que ha habido un fallo al guardar el tiempo y reajustamos el TimeStamp. El reajuste se realiza estableciendo el valor del TimeStamp de tal manera que entre el dato anterior y el que se está considerando haya una diferencia de 100ms.

2. Para cada una de las tres coordenadas que posee el fichero X, Y, Z se calculan ventanas de 8000 ms y un solapamiento de 1600ms.

3. Se calculan la media, la desviación estándar el mínimo y el máximo de cada ventana.

4. Se añaden características de identificación o atributos que definen la toma de datos.

Las características que nos permiten diferenciar e identificar a los usuarios y que se van añadiendo al dataframe son las siguientes:

- Numero de ventana: Indica el número de ventana que se ha procesado
- Conjunto: Indica el conjunto al que pertenece el fichero, TFG o TFM
- Usuario: Indica el nombre del usuario
- Marca: Indica la marca del dispositivo, Moto o Micro
- Sensor: Indica el sensor, ACC o GYR
- Sesión: Indica la sesión captada
- Muestra: Indica la muestra

Por último, la función devuelve el Dataframe con el usuario procesado. En la siguiente figura se muestra la implementación de este método:

La segunda subtarea, en la preparación y acondicionamiento de los datos, es la *Creación de función capaz de procesar los directorios TFG y TFM*. Esta tarea, tal como su nombre indica, consiste en crear una función capaz de procesar los directorios TFG y TFM, directorios que contienen diferentes números de usuarios y datos.

La operación de *process_TFG y process_TFM* son las encargadas de realizar esta función. Esta función procesa los directorios de ambos conjuntos de usuarios añadiendo al Dataframe global la información de cada usuario en particular.

La estructura de la función es la siguiente:

1. Se recorren todos los ficheros del directorio. Dependiendo de la función este puede ser TFG o TFM

2. Para cada fichero del directorio, se añaden columnas extra

3. Para cada fichero del directorio, se utiliza la función *preprocesar*

Las columnas que se van añadiendo al dataframe para cada usuario perteneciente al directorio de ejecución son las siguientes:

- Conjunto: Si la función es *process_TFG se añade* TFG y si la función es *process_TFM* se añade TFM
- Usuario: Indica el nombre del usuario añadiendo antes del nombre TFG o TFM Si la función es *process_TFG o* la función es *process_TFM* respectivamente.
- Marca: Indica la marca del dispositivo, Moto o Micro
- Sensor: Indica el sensor, ACC o GYR
- Sesión: Indica la sesión captada
- Muestra: Indica la muestra

Las funciones *process_TFG* y *process_TFM* tienen una estructura similar a diferencia de pequeños cambios como el path al directorio o el nombre de las columnas que se añaden al dataframe tal como se acaba de mencionar.

La tercera subtarea, en la preparación y acondicionamiento de los datos, es la *Creación de función capaz de guardar los datos ya procesados.* La importancia de esta tarea se basa en aportar rapidez de ejecución al proyecto si disponemos de los datos procesados.

Uno de los inconvenientes generales de los modelos de IA es el procesamiento de los datos, ya que estos suelen ser costosos desde un punto de vista temporal, lo mismo sucede en nuestro proyecto.

La función *process_TFG* requiere un promedio de 5 minutos en el caso de procesar del conjunto de datos TFG que tiene 12 usuarios y la función *process_TFM* tarda alrededor de 20 minutos en procesar el conjunto de datos TFM formado por 24 usuarios. Si tuviéramos que procesar los mismos datos siempre que ejecutamos el programa, este seria inservible debido a la cantidad de tiempo que requiere el procesamiento.

Para mejora el tiempo de ejecución de ambas funciones de esta tarea, se han implementado hilos concurrentes que procesan ambos conjuntos simultáneamente. Se ha calculado y el tiempo total de ejecución después de añadir el procesamiento concurrente es de aproximadamente 10 minutos para ambos conjuntos.

Aunque hay una mejora considerable añadiendo los hilos, el coste temporal que supone el uso de estas funciones sigue siendo alto, lo que buscamos en esta primera parte del sistema es tener que realizar esta operación una sola vez, es decir, no tener que volver a procesar los conjuntos de usuarios cada vez que ejecutemos el programa. Para ello lo que se propone es una procesar una sola vez los directorios que contienen los datos en crudo y guardar el procesado del Dataframe global.

Tal como se muestra en la siguiente figura:

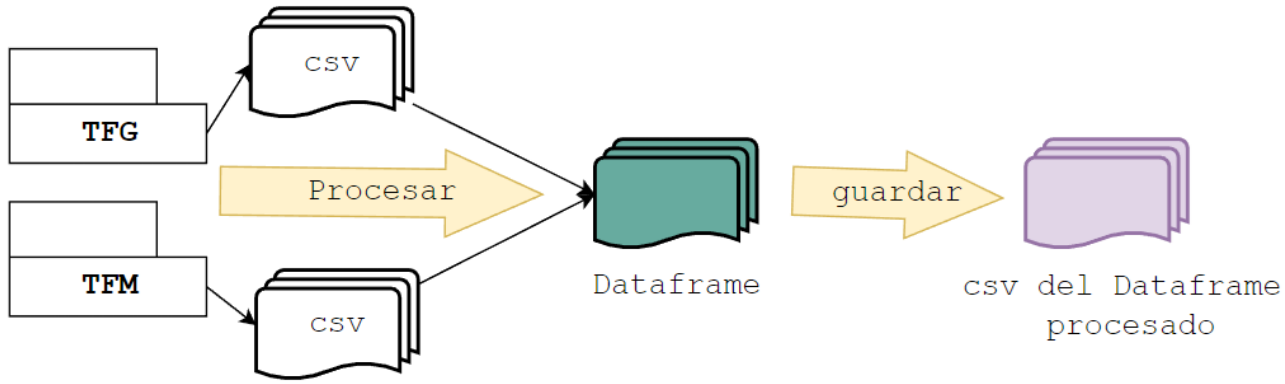


Figura 27 Procesamiento y almacenamiento de datos ponibles

En esta tarea, se ha creado una función que guarda el dataframe temporal, como fichero csv, que contienen los datos procesados de todos los usuarios, generados previamente por las funciones *process_TFG* y *process_TFM.*

La cuarta subtarea, en la preparación y acondicionamiento de los datos, es la *Creación de una función capaz de importar los datos procesados.*

Una solución para evitar el procesamiento de datos cada vez que ejecutamos el programa es importar los datos previamente guardados. Tal como se muestra en la *figura 26* una vez procesados los conjuntos no hace falta volver a hacerlo porque ya tenemos el Dataframe guardado en el sistema. Con el desarrollo de esta función las posibilidades que ofrece el programa en cuanto al tratamiento de datos se muestran en la siguiente figura.

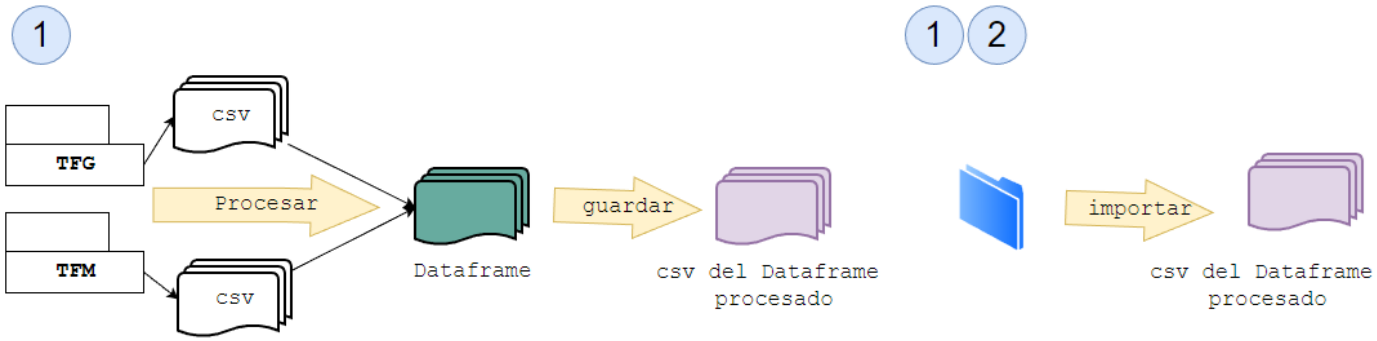


Figura 28 Secuencia de posbilidades a la hora de procesar los datos

En un primer lugar o como primera opción si no disponemos del conjunto de datos procesado, lo que hacemos es procesar ambos conjuntos de datos y guardar en un fichero los datos procesados. Posteriormente importamos dichos datos desde el sistema.

Si ya disponemos del conjunto de datos procesado de ejecuciones anteriores, en esta tarea, se ha creado una función que importa el fichero csv, que contienen los datos procesados de todos los usuarios, generados previamente por las funciones *process_TFG* y *process_TFM*, esto hace que no tengamos que esperar 25 minutos a que finalice el procesamiento de datos.

La quinta subtarea, en la preparación y acondicionamiento de los datos, es la *Creación de una función que separa los datos en entrenamiento y prueba.* Tal como su nombre lo indica, el objetivo de la tarea es crear dos conjuntos que servirán de entrada a los modelos de IA, el conjunto de entrenamiento y el conjunto de pruebas.

La función encargada de realizar esto es *load_dataset()*. Esta función recibe como parámetro el conjunto de datos procesados y devuelve el Objeto DATA que se explicara más tarde en esta misma sección. Para obtener dichos conjuntos se ha hecho lo siguiente.

Suponiendo fijo el usuario, la marca (Microsoft o Motorola) y el sensor (ACC o GYR). Cada conjunto, Entrenamiento o Prueba se ha dividido en dos grupos de usuarios, los usuarios auténticos o los impostores.

- En el conjunto de entrenamiento los usuarios auténticos son aquellos que pertenecen al conjunto de datos TFM, y tienen la muestra 1 de la sesión 1, y a estos se les asigna el valor 1. Entre los impostores del conjunto de entrenamiento se encuentran todos los usuarios del conjunto TFG, asignándoles un valor de 0.

- En el conjunto de prueba los usuarios auténticos son aquellos que pertenecen al conjunto de datos TFM, teniendo las muestras 1 y 2 de la sesión 2, y a estos se les asigna el valor 1. Entre los impostores del conjunto de entrenamiento se encuentran las muestras no usadas de del conjunto TFM, es decir aquellos que cuya sesión sea distinta de la muestra 1 de la sesión 1, asignándoles una salida con valor 0.

7.1.4. Pruebas de la solución implementada

En la sección *pruebas y verificación de la solución implementada* se analizarán los resultados de cada una de las implementaciones de las subtareas realizadas a lo largo del sprint.

La primera subtarea, perteneciente a la épica: preparación y acondicionamiento de los datos, es la *Creación de una función capaz de procesar un fichero de datos ponibles.* Como se ha comentado, el objetivo de esta tarea, es diseñar una función capaz de procesar los datos en crudos procedentes de los ficheros csv de cada usuario. En si no es una función que se llame para ejecutar un solo fichero csv así que la verificación de su funcionamiento se realizara junto con la siguiente tarea.

La segunda subtarea, perteneciente a la épica: preparación y acondicionamiento de los datos, es la *Creación de función capaz de procesar los directorios TFG y TFM*. Como se ha comentado, el objetivo de esta tarea, es diseñar una función capaz de procesar los directorios TFG y TFM, directorios que contienen diferentes números de usuarios y datos.

Las funciones desarrolladas a lo largo de esta tarea son las siguientes: *process_TFG* y *process_TFM*. Estas funciones utilizan también la función *preprocesar()* creada en la primera subtarea de esta épica. Por tanto, la validación de esta subtarea sirve para la anterior.

A continuación, se muestra la ejecución de la función *process_TFG*:

```
Procesando  Micro_ACC_usuario10_1_2_1.csv
Procesando  Micro_GYR_usuario10_1_2_1.csv
Procesando  Micro_ACC_usuario10_1_2_2.csv
Procesando  Micro_GYR_usuario10_1_2_2.csv
Procesando  Moto_GYR_usuario10_1_1_1.csv
Procesando  Moto_ACC_usuario10_1_1_1.csv
Procesando  Moto_ACC_usuario10_1_1_2.csv
Procesando  Moto_GYR_usuario10_1_1_2.csv
Procesando  Moto_GYR_usuario10_2_1_1.csv
Procesando  Moto_ACC_usuario10_2_1_1.csv
Procesando  Moto_GYR_usuario10_2_1_2.csv
Procesando  Moto_ACC_usuario10_2_1_2.csv
Procesando  Micro_GYR_usuario10_2_2_1.csv
Procesando  Micro_ACC_usuario10_2_2_1.csv
Procesando  Micro_ACC_usuario10_2_2_2.csv
Procesando  Micro_GYR_usuario10_2_2_2.csv
Procesando  Moto_ACC_usuario14_1_1_1.csv
Procesando  Moto_GYR_usuario14_1_1_1.csv
Procesando  Moto_GYR_usuario14_1_1_2.csv
Procesando  Moto_ACC_usuario14_1_1_2.csv
Procesando  Micro_ACC_usuario14_1_2_1.csv
Procesando  Micro_GYR_usuario14_1_2_1.csv
Procesando  Micro_ACC_usuario14_1_2_2.csv
Procesando  Micro_GYR_usuario14_1_2_2.csv
Procesando  Moto_GYR_usuario14_2_1_1.csv
Procesando  Moto_ACC_usuario14_2_1_1.csv
Procesando  Moto_ACC_usuario14_2_1_2.csv
Procesando  Moto_GYR_usuario14_2_1_2.csv
Procesando  Micro_ACC_usuario14_2_2_1.csv
Procesando  Micro_GYR_usuario14_2_2_1.csv
Procesando  Micro_GYR_usuario14_2_2_2.csv
```

Figura 29 Ejecución de la función process TFG

Como podemos observar, se van procesando todos los usuarios del conjunto TFG hasta completar los 12 usuarios de los que dispone el directorio. Para comprobar que el procesamiento es efectivo, después de ejecutar la función *process_TFG* se imprime TodoFeat que es el dataframe donde se guardan los usuarios procesados con sus características. Tal como se puede observar en la *figura 29* el procesado realizado por la función es efectivo:

```
print(TodoFeatDF)
```

```
    mean_dato1  std_dato1  min_dato1  max_dato1  mean_dato2  std_dato2  \
0     0.816318   0.267356   0.379639   1.379639    0.435247   0.175016
1     0.907680   0.258880   0.469727   1.506592    0.360255   0.106517
2     0.939921   0.218322   0.527100   1.349365    0.345844   0.106537
3     0.927862   0.195458   0.567871   1.312744    0.363365   0.117283
4     0.937915   0.208003   0.528320   1.361572    0.351154   0.092625
..         ...        ...        ...        ...         ...        ...
15   -0.970366   0.361616  -1.863281  -0.074707   -0.322386   0.150492
16   -0.855725   0.625959  -2.725586   1.286621   -0.331592   0.242152
17   -0.773606   0.649351  -2.725586   1.286621   -0.413110   0.302288
18   -0.899191   0.343070  -2.061035  -0.332520   -0.471957   0.207273
19   -0.659644   0.494670  -2.032227   0.357910   -0.581073   0.240639

    min_dato2  max_dato2  mean_dato3  std_dato3  min_dato3  max_dato3  \
0   -0.214111   0.872803    0.391184   0.154942   0.065918   0.882324
1    0.141113   0.621826    0.331008   0.119450   0.089355   0.630615
2    0.125488   0.564453    0.327757   0.114256   0.078857   0.550293
3    0.036377   0.627686    0.278523   0.112174  -0.011230   0.536621
4    0.175781   0.536621    0.288780   0.105894   0.072754   0.554688
..        ...        ...         ...        ...        ...        ...
15  -0.825195  -0.052246    0.372872   0.241254   0.022461   0.982910
16  -1.723633   0.146484    0.364679   0.301259  -1.144531   0.930664
17  -1.723633   0.146484    0.238444   0.361418  -1.144531   0.954590
18  -0.911621   0.009766    0.171405   0.199837  -0.132324   0.677734
19  -1.145508  -0.098633   -0.123951   0.405207  -1.501953   0.717285

    NumVent Conjunto       Usuario  Marca Sensor  Sesion  Muestra
0         0      TFG  TFGusuario10  Micro    ACC       1        1
1         1      TFG  TFGusuario10  Micro    ACC       1        1
2         2      TFG  TFGusuario10  Micro    ACC       1        1
3         3      TFG  TFGusuario10  Micro    ACC       1        1
4         4      TFG  TFGusuario10  Micro    ACC       1        1
..      ...      ...           ...    ...    ...     ...      ...
15       15      TFG   TFGusuario2   Moto    ACC       2        2
16       16      TFG   TFGusuario2   Moto    ACC       2        2
17       17      TFG   TFGusuario2   Moto    ACC       2        2
18       18      TFG   TFGusuario2   Moto    ACC       2        2
19       19      TFG   TFGusuario2   Moto    ACC       2        2

[4400 rows x 19 columns]
```

Figura 30 Dataframe después del preprocesado del conjunto TFG

La tercera subtarea, perteneciente a la épica: preparación y acondicionamiento de los datos, es la *Creación de función capaz de guardar los datos ya procesados.* Como se ha comentado, el objetivo de esta tarea, es diseñar una función capaz de guardar el dataframe temporal generado por la tarea anterior en formato csv.

Tal como se observa en la figura XX, al ejecutar la función obtenemos el fichero TodoFeat.csv, que tiene las siguientes columnas, que las podemos separar en dos grandes grupos:

- Columnas de coordenadas: Estas columnas tienen información sobre las coordenadas X,Y,Z entre estas columnas se encuentran: *mean_dato1 ,std_dato1min_dato1,max_dato1 mean_dato2, std_dato2, min_dato2, max_dato2 mean_dato3, std_dato3, min_dato3 y max_dato3*.

Min, max, mean y std hacen referencia al mínimo al máximo a la media y a la desviación estándar respectivamente

- Características del usuario: Estas columnas son características del usuario entre estas se numeran NumVent, Conjunto, Usuario, Marca, Sensor, Sesión y Muestra.

A	B	C	D	E	F	G	H	I	J
	mean_dato1	std_dato1	min_dato1	max_dato1	mean_dato2	std_dato2	min_dato2	max_dato2	mean_dato3
0	0.8163180361	0.2673556556	0.379639	1.379639	0.4352468675	0.1750162568	-0.214111	0.872803	0.3911838675
1	0.9076795783	0.2588796228	0.469727	1.506592	0.3602545301	0.1065171168	0.141113	0.621826	0.3310076145
2	0.9399208554	0.2183215109	0.5271	1.349365	0.3458442651	0.1065365333	0.125488	0.564453	0.3277573494
3	0.9278624024	0.1954581774	0.567871	1.312744	0.3633645488	0.1172826821	0.036377	0.627686	0.2785227073
4	0.9379147229	0.2080034687	0.52832	1.361572	0.3511536265	0.09262545309	0.175781	0.536621	0.2887800843
5	0.9448007229	0.2216058183	0.557373	1.362549	0.3726703373	0.1186946533	0.148682	0.720703	0.2709078675
6	0.9307309405	0.2454969788	0.510254	1.370605	0.3568085595	0.09900781842	0.186768	0.571045	0.2804565238
7	0.9504482651	0.2378077616	0.586914	1.386963	0.3558334458	0.1002080797	0.166748	0.571045	0.2703636747
8	0.9243134217	0.2554068525	0.533691	1.421631	0.3415851084	0.08073840907	0.177002	0.540039	0.2770349277
9	0.9266695663	0.2338629834	0.587646	1.410889	0.3674080843	0.1051033324	0.139404	0.546875	0.2802057952
10	0.9046116145	0.2543568621	0.472412	1.42041	0.3595220843	0.1061409861	0.17749	0.726074	0.2837884699
11	0.8941076627	0.2533743428	0.500732	1.401611	0.381821241	0.1262132019	0.115479	0.695312	0.2642042289
12	0.9331083855	0.2344029655	0.512207	1.407227	0.3597132892	0.1273634604	0.145996	0.730713	0.2525649157
13	0.9412033133	0.2300934995	0.515869	1.357422	0.3747764217	0.1249572118	0.155273	0.69751	0.2357603855
14	0.9325348313	0.2327585452	0.498779	1.357422	0.3527832048	0.1094343525	0.161865	0.588379	0.2821148072
15	0.9440976988	0.2222199246	0.498779	1.304688	0.3554863614	0.1010112468	0.205566	0.603027	0.2598127108
16	0.9639819277	0.2184425653	0.544678	1.468994	0.3808623133	0.1276683182	0.12915	0.633057	0.2437435181
17	0.956349439	0.1944648344	0.565186	1.321533	0.3706977195	0.1295975457	0.124023	0.616699	0.2453256098
18	0.926948988	0.2282883249	0.484619	1.34082	0.371331988	0.1150068813	0.175049	0.799316	0.2387018916
19	0.9259665904	0.2345067741	0.498047	1.367432	0.3786973735	0.1102049098	0.220703	0.799316	0.2255270602
20	0.8990435181	0.2583739807	0.251221	1.318848	0.4451330482	0.1661064573	0.193359	1.027832	0.1499141084

Tabla 9 Datos ponibles procesados

7.2. SPRINT 2: Modelo Inicial

7.2.1. Objetivo

La creación de los modelos para la plataforma de pruebas es esencial para el funcionamiento de la aplicación. Antes de crear los modelos seleccionables hay que crear modelos *vainilla* o modelos básicos para poder entender los parámetros y el funcionamiento de estos. Este Sprint tiene como objetivo la creación de un modelo MLP y un modelo CNN capaces de funcionar con los datos proporcionados.

7.2.2. Análisis de objetivos

Los objetivos en Scrum, son metas tangibles redactadas en conjunto por el equipo que están sujetas a la duración del sprint. Por tanto, el objetivo de este segundo sprint al igual que el de los anteriores es completar las 3 historias de usuario o épicas que se detallan a continuación:

- Sprint 2: Modelo Inicial
    - Épica 1: Creación de un modelo MLP inicial
    - Épica 2: Creación de un modelo CNN inicial
    - Épica 3: Documentación

Como se puede observar de la descripción anterior, **como primera épica** de este sprint, tenemos la *creación de un modelo MLP*. Esto nos permite entender los parámetros del modelo MLP y probar que los datos que se han procesado resultan de utilidad. En la *sección 2.1.1 titulada perceptrón multicapa en la práctica* se analizan con detenimiento los posibles parámetros de este modelo. Como la complejidad de esta épica no es elevada no dispondrá de subtareas.

Como **segunda épica** tenemos la *creación de un modelo de redes convolucionales* (CNN), esto nos permite entender los parámetros del modelo CNN y probar que los datos que se han procesado resultan de utilidad. Esta épica tiene mayor dificultad que la anterior, ya que previamente a este proyecto no se había trabajado con redes CNN, además el modelo CNN es un modelo complejo con un mayor número de parámetros en comparación con el MLP, y es fácil que produzca errores. En la sección 2.2.4 titulada *redes CNN en la práctica* se analizan con detenimiento los posibles parámetros de este modelo.

Por último, como ***tercera épica***, al igual que sucede en cada sprint tenemos como historia la documentación. Esta tarea consiste en redactar y documentar lo realizado a lo largo de este Sprint.

A continuación, se analiza cómo se ha realizado el desarrollo de las dos épicas de este sprint, *creación de un modelo MLP y creación de un modelo CNN.*

7.2.3. Desarrollo e implementación

La primera épica, es la *creación de un modelo MLP.* Esta épica, tal como su nombre indica, consiste en crear un modelo perceptrón multicapa MLP capaz de dar solución a la clasificación de usuarios con los datos procesados de los ponibles.

Es posible que el nombre que se ha dado a las funciones que se van a mencionar en este sprint, haya cambiado o hayan desaparecido en los sprints siguientes debido a que las funciones que desarrollan los modelos se han hecho más complejas. El código generado en este sprint sirve como prueba de concepto.

La función que se encarga de crear el modelo MLP en este sprint se denomina *model_mlp()*. Esta función es un "todo en uno", se encarga de preparar los datos que recibe el modelo separándolos en conjuntos de entrenamiento y prueba, crea el modelo y lo entrena y por último imprime los resultados.

Los parámetros utilizados en la creación de este modelo han sido los proporcionados por defecto por la función *MLPClassifier* de sklearn. A continuación se hace un *fit()* con los conjuntos entrenamiento y prueba proporcionados por la función *load_data()* perteneciente al sprint 1. Y por último se hace un *predict* con el conjunto de prueba. La función *model_mlp()* devuelve el porcentaje de acierto o accuraccy_score entre la predicción realizada sobre el conjunto de prueba y el actual conjunto de prueba.

La segunda épica, es la *creación de un modelo CNN.* Esta épica, tal como su nombre indica, consiste en crear un modelo de red convolucional capaz de dar solución a la clasificación de usuarios con los datos procesados de los ponibles.

Aunque no se ha añadido como subtareas de esta épica, se ha dedicado tiempo en comprender el funcionamiento de las redes neuronales CNN y sus características. También se han realizado numerosas pruebas para crear la estructura base del modelo y obtener un resultado aceptable en la ejecución de este.

Para la creación del modelo CNN hemos utilizado Keras que se basa en TensorFlow. Keras se especializa en modelos de aprendizaje profundo como son las redes CNN. A continuación, se explican conceptos importantes que hemos utilizado en la implementación de este modelo. Este api permite dos implementaciones de las redes neuronales convolucionales que explicaremos a continuación:

- *Sequential*: Esta forma de crear redes neuronales CNN es la más sencilla, y es la que utilizaremos en este proyecto. El modelo se construye capa a capa y todas las neuronas de una capa tienen como entrada la salida de todas las neuronas de la capa anterior. Un ejemplo de lo explicado con anterioridad lo podemos observar en la *figura 30*.

- *Functional:* La forma en la que se crea el modelo es más flexible que la *Sequential.* Añade complejidad en la forma en la que se organizan las capas y las neuronas, estas se pueden ramificar o se pueden compartir capas. Como podemos observar en la *figura 31*, esto nos permite haber creado 2 redes neuronales diferentes, lo que nos permitiría predecir dos salidas en una sola red.

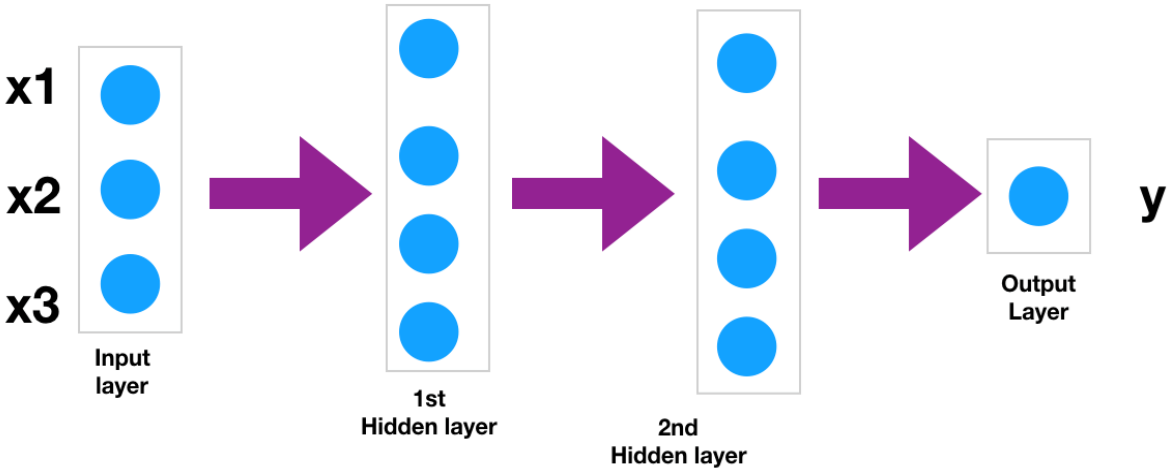


Figura 31 Modelo CNN Sequential

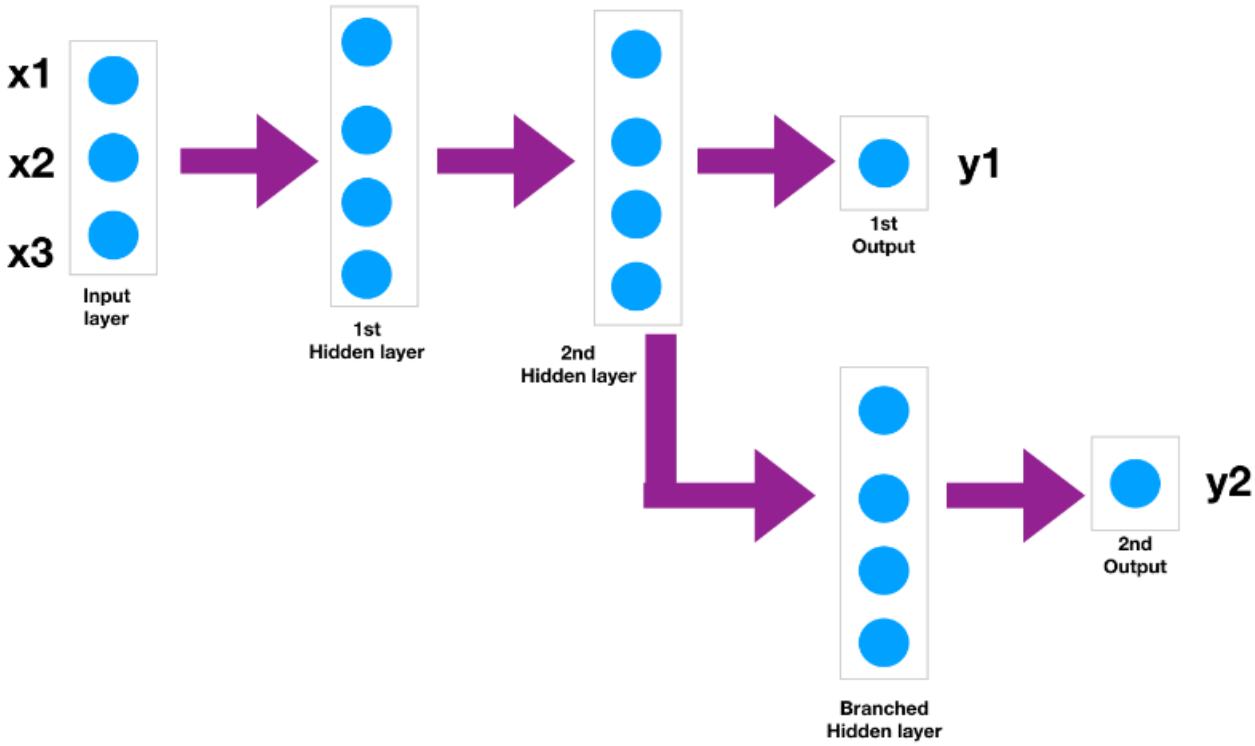


Figura 32 Modelo CNN Functional

El siguiente paso en la creación de nuestro modelo CNN una vez elegida la estructura *Sequential* es añadir o apilar capas. La función que nos permite añadir capas a nuestro modelo se denomina *add*, esta tiene como argumento la capa siguiente a añadir. La estructura que tendrá nuestra modelo CNN es la que se muestra en la *figura 32.* Hay que reconocer que a lo largo de la épica se han añadido las capas referenciadas poco a poco, y que todas las capas mostradas en la figura son las capas resultantes de la finalización de esta épica.

El modelo lo componen las siguientes capas que van en orden y están totalmente conectadas, es decir la salida de una capa es la entrada de la siguiente:

1. Conv1D: Esta es la primera capa de nuestro modelo sequencial CNN, se utiliza para tratar datos de naturaleza temporal.

2. MaxPooling1D: Como se ha explicado en capítulos anteriores la capa de MaxPooling selecciona el mayor valor de una ventana de tamaño especifico.

3. Dropout: El *dropout* es una técnica de regularización para reducir el sobreajuste en las redes neuronales al evitar adaptaciones complejas en los datos de entrenamiento

4. Flatten : El objetivo de esta capa es convertir los datos que recibe en una matriz unidimensional

5. Dense: Es una capa totalmente conectada, es decir recibe información de cada una de las neuronas de la capa previa. Como es la que dará el resultado a nuestro modelo, a esta se la ha dado tamaño 1 y será la ultima capa de nuestro modelo.

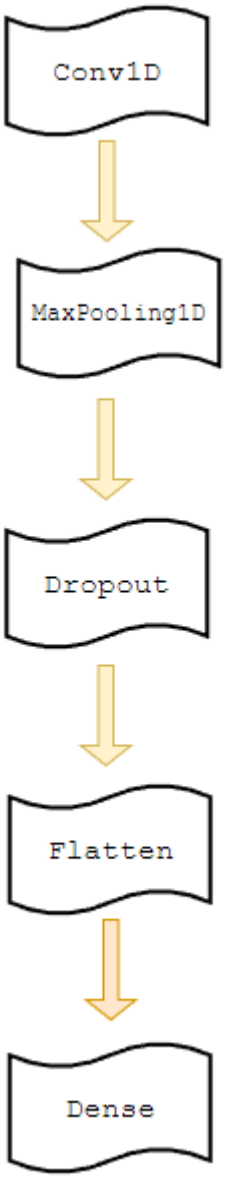


Figura 33 Estructura del modelo CNN

Después de añadir las capas compilamos el modelo. La compilación realizada tiene los parámetros preestablecidos por defecto, entre estos se encuentran la función de perdida '*binary_crossentropy*' como optimizador predeterminado esta '*Adam'* y como métricas deseamos el *accuracy* o acierto.

A continuación, entrenamos el modelo mediante la función *fit*, esta recibe como parámetros los conjuntos de entrenamiento y los conjuntos de prueba. Por último, imprimimos el acierto de la ejecución del modelo.

7.2.4. Pruebas de la solución implementada

En la sección *pruebas y verificación de la solución implementada* se analizarán los resultados de cada una de las implementaciones de las épicas realizadas a lo largo del sprint.

La primera épica, *creación de un modelo MLP.* Tiene como objetivo crear un modelo perceptrón multicapa MLP tal como se ha explicado en la sección de implementación de este mismo capítulo. Las pruebas de la solución implementada en esta épica consistirían en la ejecución del modelo MLP creado.

A continuación, se muestra la ejecución del modelo generado:

```
<---------- MODELO MLP ---------->

<========== EJECUCION 1==========
>#1: 76.185

<========== RESULTADOS <==========>
[76.18547681539808]
Accuracy: 76.185% (+/-0.000)
```

Figura 34 Ejecucion modelo MLP inicial

La segunda épica, de este sprint es la *creación de un modelo CNN* tal como se ha comentado en la sección de implementación de este mismo capítulo. Las pruebas de la solución implementada de esta épica consistirían en la ejecución del modelo CNN creado.

A continuación, se muestra la ejecución del modelo generado:

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Conv_1 (Conv1D)             (None, 10, 64)            256

 MaxPool_1 (MaxPooling1D)    (None, 5, 64)             0

 Dropout (Dropout)           (None, 5, 64)             0

 Flatten (Flatten)           (None, 320)               0

 Dense_Last (Dense)          (None, 1)                 321

=================================================================
Total params: 577
Trainable params: 577
Non-trainable params: 0
```

Figura 35 Estructura de capas del modelo CNN inicial ejecutado

```
Accuracy: 98.766% [98.76640439033508]
```

Como hemos visto con anterioridad el modelo propuesto en este sprint es un modelo *sequential(),* que se compone de 5 capas o layers, en primer lugar tenemos una capa *Conv1D*, que tiene 64 neuronas por defecto.

A continuación, como segunda capa tenemos un MaxPool que selecciona el máximo valor de la ventana, dos capas Intermedias una *Dropout* y una *Flatten*, que intenta reducir el reajuste y que transforma la salida en una matriz unidimensional respectivamente.

Por último, tenemos una capa *Dense* que nos ofrece la solución a nuestro modelo. Aunque el resultado de este modelo es muy positivo, cabe destacar que el porcentaje de acierto no es relevante en este proyecto.

7.3. SPRINT 3: Plataforma de Pruebas

Este sprint es semejante al anterior, ya que se utilizarán como base los modelos generados en el Sprint previo y las funciones que procesan los datos en crudo, generadas en el Sprint 1.

7.3.1. Objetivo

Una vez creados los modelos "vainilla", disponemos de una base para poder crear nuestra plataforma de pruebas. Este Sprint tiene como objetivo la creación de modelos seleccionables que permiten a los usuarios elegir los parámetros con los que desean entrenar su modelo, también se refinaran las funciones existentes para adaptarlas al objetivo de este Sprint.

7.3.2. Análisis de objetivos

Los objetivos en Scrum, son metas tangibles redactadas en conjunto por el equipo que están sujetas a la duración del sprint. Por tanto, el objetivo de este tercer sprint al igual que el de los anteriores es completar las 3 historias de usuario o épicas que se detallan a continuación:

- Sprint 3: Plataforma de Pruebas
    - Épica 1: Creación de un marco de pruebas para MLP
    - Épica 2: Creación de un marco de pruebas para CNN
    - Épica 3: Documentación

Como se puede observar de la descripción anterior, como **primera épica** de este sprint, tenemos la *creación de un marco de pruebas para MLP*, esta épica consiste en modificar el modelo MLP creado en el Sprint anterior y permitir seleccionar al usuario entre ciertos parámetros de interés. En la *sección 2.1.1 titulada perceptrón multicapa en la práctica* se analizan con detenimiento todos los posibles parámetros de este modelo.

Esta tarea no es directa, al igual que sucedía en el sprint anterior. Al tener una complejidad mayor podemos descomponer la implementación de esta épica en varias subtareas que se detallan a continuación:

- Creación de Objetos para el modelo MLP.
- Creación de una función que ejecuta el experimento e imprime los resultados.

- Adaptación del modelo MLP inicial a un marco de pruebas.

Como **segunda épica** de este sprint, tenemos la *creación de un marco de pruebas para CNN*, esta épica consiste en modificar el modelo CNN creado en el Sprint anterior y permitir seleccionar al usuario entre ciertos parámetros de interés. En la *sección 2.2.4 titulada redes CNN en la práctica* se analizan con detenimiento todos los posibles parámetros de este modelo.

Cabe destacar que la implementación de está épica junto con el procesamiento de datos han sido una de las más complejas del TFG debido al error de los parámetros y el tiempo empleado en conseguir un modelo estable.

Esta tarea no es directa, al igual que sucedía en el sprint anterior. Al tener una complejidad mayor podemos descomponer la implementación de esta épica en varias subtareas que se detallan a continuación:

- Creación de Objetos para el modelo CNN.
- Creación de funciones que ejecuta el experimento, imprime los resultados y muestra análisis.
- Adaptación del modelo CNN inicial a un marco de pruebas.

A continuación, se analiza cómo se ha realizado el desarrollo de las dos épicas de este sprint, *creación de un marco de pruebas para MLP y creación de un marco de pruebas para CNN.*

7.3.3. Desarrollo e implementación

La **primera épica**, *creación de un marco de pruebas para MLP*. Contiene 3 subtareas tal como se ha mostrado en el análisis realizado con anterioridad. La primera de las subtareas, de esta épica es la creación de objetos para el modelo MLP.

En esta primera subtarea común a las dos épicas de este sprint, tiene como objetivo la creación de objetos que nos hagan más fácil el manejo de datos y parámetros que recibimos por parte del usuario.

Al ser un modelo sencillo, el único objeto que necesitaremos para la implementación del marco de pruebas es el objeto *Data* que se utilizara posteriormente también para el modelo *CNN*. Este objeto contendrá los diferentes conjuntos de entrenamiento y prueba y serán entrada tanto del modelo MLP planteado en esta épica como para las redes CNN. El objeto data tiene los siguientes atributos:

- trainx: contiene el conjunto de datos de entrenamiento.
- trainy: contiene la salida del conjunto de entrenamiento.
- testx: atributo que contiene el conjunto de prueba.
- testy: contiene la salida del conjunto de prueba.

Las operaciones del objeto Data son las siguientes:

- get_trainx: devuelve el conjunto de datos de entrenamiento.
- get_trainy: devuelve la salida del conjunto de entrenamiento.
- get_testx: devuelve conjunto de prueba.
- get_testy: devuelve la salida del conjunto de prueba.

El objeto Data no solo se utiliza como parámetro para los modelos IA de nuestro sistema. Tal como se ha mencionado en la implementación de la creación del conjunto de entrenamiento y de prueba, la función *load_data()* devuelve un objeto de tipo Data separando los conjuntos de entrenamiento y prueba.

La segunda de las subtareas, de esta épica, es la *creación de una función que ejecuta el experimento e imprime los resultados*. El modelo MLP de nuestro marco de pruebas consta de dos funciones, una función que se encarga de ejecutar el modelo con los parámetros seleccionado por los usuarios y otra función que llama a esta última añadiendo el número de veces que queremos ejecutar el modelo e imprime el resultado de las ejecuciones de este.

En esta segunda subtarea analizaremos la función encargada de llamar al método que ejecuta el modelo e imprime los resultados de este. La función *run_experiment_mlp* recibe como parámetros:

- *Data*: Objeto que contiene los conjuntos de prueba y de entrenamiento de los datos ponibles.
- *Parametros*: parámetros seleccionados por el usuario para este modelo, se mencionarán en la subtarea siguiente.

Además de los parámetros seleccionables del propio modelo, existen parámetros globales que son independientes del modelo. Entre estos parámetros tenemos:

- *repeticiones*: indica el número de veces que deseamos que se ejecute el modelo.

Esta función tiene como objetivo llamar a la función que ejecuta el modelo MLP, con los parámetros seleccionados por el usuario, tantas veces como repeticiones se haya indicado. Imprime los resultados de las diferentes ejecuciones, calculando la media de estas ejecuciones y la desviación estándar de estas.

Por ejemplo, si seleccionamos dos repeticiones, ejecutaremos dos veces el modelo MLP, obteniendo así dos porcentajes de acierto, la función imprime estos resultados, imprime la media de estos y su desviación estándar. Un ejemplo de esto se mostrará en la implementación de esta subtarea.

La tercera de las subtareas, de esta épica, es la *adaptación del modelo MLP inicial a un marco de pruebas.* En esta subtarea se permitirá al usuario seleccionar parámetros para entrenar el modelo MLP, como se ha comentado con anterioridad, run*_experiment_mlp* llama a *evaluate_model_mlp* que es la encargada de crear compilar y ejecutar el modelo MLP.

De todos los parámetros de los que disponen las MLP explicados con detenimiento en la sección 2.2.1 titulada Perceptrón Multicapa en la Práctica, para el marco de pruebas de este proyecto se han elegido los siguientes parámetros seleccionables, que servirán como entrada de nuestra función:

- *hidden_layer_sizes:* Es el primer parámetro seleccionable de nuestro modelo, este hace referencia al número de capas ocultas que deseamos. Cada capa oculta que elijamos tiene por defecto 100 neuronas.

- *max_iter:* Es el segundo parámetro seleccionable de nuestro modelo, hace referencia al máximo número de iteraciones. Si no seleccionamos nada el valor que tiene por defecto de 10000 iteraciones.

La **segunda épica**, *creación de un marco de pruebas para CNN*. Contiene 3 subtareas tal como se ha mostrado en el análisis realizado con anterioridad. La primera de las subtareas, de esta épica es la creación de objetos para el modelo CNN.

En esta primera subtarea común a las dos épicas de este sprint, tiene como objetivo la creación de objetos que nos hagan más fácil el manejo de datos y parámetros que recibimos por parte del usuario.

La creación del modelo *CNN* es más compleja que la de un modelo *MLP* debido a la cantidad de parámetros que este tiene.

Los objetos que necesitamos para construir el modelo son los siguientes:

- *Capa_1D*: Objeto que nos permite tratar listas de *capas 1D,* a su vez es un parámetro del objeto Modelo

- *Capa_dense*: Objeto que nos permite tratar listas de *capas dense,* a su vez es un parámetro del objeto Modelo

- *Data*: Objeto que nos permite tratar los conjuntos de entrenamiento y prueba

- *Modelo:* Objeto que contiene los parámetros del modelo *CNN*

- *Options*: Objeto que contiene las opciones de ejecución del modelo *CNN*

A continuación, se analiza la implementación de cada objeto y sus valores predeterminados.

En primer lugar, analizaremos el objeto Capa_1D, este objeto permitirá la creación de un conjunto Conv1D y MaxPooling1D. Una instancia de Capa_1D contiene una capa Conv1D y una capa MaxPooling1D tal como se muestra en la siguiente figura:

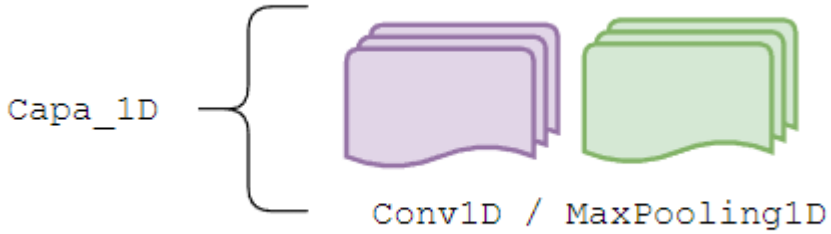


Figura 36 Estructura de Capa_1D

Este objeto permite la creación de conjuntos de capas Conv1D y Maxpooling1D dados los valores *tam_conv1D y tam_maxpooling1D* respectivamente. El objeto Capa_1D tiene los siguientes atributos:

- *tam_conv1D*: indica el tamaño de tam_conv1D. Si no se indica nada el valor por defecto es 64.
- *tam_maxpooling1D*: indica el tamaño de maxpooling1D. Si no se indica nada el valor por defecto es 2.

Las operaciones del objeto Capas_1D son las siguientes:

- *get_tam_conv1D*: devuelve el tamaño de Conv1D.
- *get_ tam_conv1D*: devuelve el tamaño de Conv1D.

Otro objeto importante para nuestro modelo CNN es Capa_dense este objeto permite la creación de capas dense. Una instancia de Capa_dense tal como su nombre indica contiene una capa dense. El objeto Capa_dense tiene los siguientes atributos:

- tam: número de neuronas de la capa dense. Si no se indica nada el valor por defecto es 128.

Las operaciones del objeto Capa_dense son las siguientes:

- get_tam: devuelvo el valor del atributo tam que indica el número de neuronas de la capa dense.

Un objeto que se utiliza en este modelo al igual que en el modelo MLP es el objeto Data. El análisis de la implementación del objeto Data se ha mencionado en la épica anterior, así que se no se reiterara información.

El objeto más importante de esta épica es Modelo, este objeto contiene información con respecto a los parámetros que al usuario se le permite seleccionar. Este objeto utiliza también los objetos anteriormente mencionados, Capa_1D y Capa_dense a parte de otros atributos que se muestran a continuación.

- *list_capas_1D*: es una lista que contendrá conjuntos de objetos Capa_1D esto permite la creación de múltiples capas diferentes (Conv1D y Maxpooling1D) explicadas con anterioridad. La figura 35 muestra como está organizado List_capas_1D

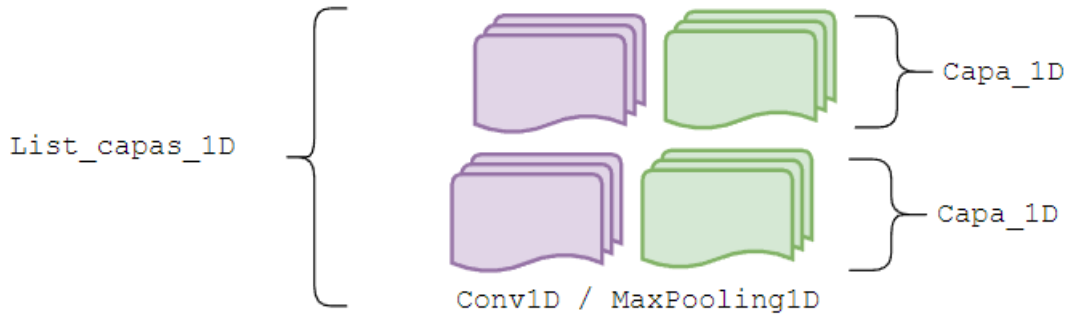


Figura 37 Estructura de List_capas_1D

- *dropout*: Si es seleccionada esta opción añade una capa *dropout*. Si no se especifica esta opción esta seleccionada por defecto.

- *flatten*: Si es seleccionada esta opción añade una capa *flatten*. Si no se especifica esta opción esta seleccionada por defecto.

- *list_capas_dense*: lista que contiene las capas dense del modelo. Por defecto contiene 1 capa dense a parte de la final que no la contamos como parámetro.

Las operaciones del objeto Modelo son las siguientes:

- *add_capa_1D:* añade una Capa_1D a la lista de capas 1D, se tiene que indicar el *tam_conv1D y tam_maxpooling1D* respectivamente.

- *add_capa_dense:* añade una Capa_dense a la lista de capas dense.

- *get_dropout:* se obtiene el valor del atributo *dropout*.

- *get_flatten:* se obtiene el valor del atributo *flatten*.

- *get_lista_capa_1D:* se obtiene el valor del atributo *list_capas_1D.*

- *get_lista_capa_dense:* se obtiene el valor del atributo *list_capas_dense.*

Además de los parámetros seleccionables del propio modelo, existen parámetros que afectan a diferentes capas de forma global, estas se han nombrado como *Options* dando nombre al objeto que las compone. Entre los atributos del objeto *Options* tenemos:

- *n_kernel*: tamaño del kernel, se utiliza para extraer las características de las imágenes. Si no se indica nada el valor por defecto es 3.

- *epochs:* número de épocas de entrenamiento. Si no se indica nada el valor por defecto es 15.

- *batch_size*: tamaño ventana o cantidad de muestras que procesa el modelo por época. Si no se indica nada el valor por defecto es 32.

La segunda de las subtareas, de esta épica, es la *creación de funciones que ejecuta el experimento, imprime los resultados y muestra análisis.* El modelo CNN de nuestro marco de pruebas consta de varas funciones que se han implementado durante esta épica y que se detallan a continuación:

- *run_experiment_cnn:* Esta función encargada ejecutar el modelo el número de veces que indique el usuario, hace llamadas a *evaluate_model_cnn,* para ejecutar el modelo y a *summarize_results* para imprimir resultados

- *evaluate_model_cnn:* Esta función es la encargada de crear, compilar y ejecutar el modelo, se mencionará con más detalle en la tercera subtarea.

- *summarize_results:* Esta función se encarga de calcular la media y la desviación estándar de los resultados de los modelos.

- *acierto_y_confusion_matrix:* Esta función se encarga de calcular entre otras cosas el acierto mediante la matriz de confusión y calcula la matriz de confusión.

- *pintar_grafica_epoch:* Esta función se encarga de pintar una gráfica que analiza la relación de los epoch con el porcentaje de acierto de este. Normalmente a medida que van aumentando los epoch, el porcentaje de acierto del modelo es mayor.

De todas las funciones anteriores, analizaremos run_*experiment_cnn* que es una de las más importantes. La función *run_experiment_cnn* recibe como parámetros:

- *Data*: Objeto que contiene los conjuntos de prueba y de entrenamiento de los datos ponibles.

- *Model*: parámetros seleccionados por el usuario para este modelo, se ha mencionado con detalle en la subtarea anterior.

- Options: parámetros que afectan a diferentes capas de forma global, se ha mencionado con detalle en la subtarea anterior.

Además de los parámetros seleccionables del propio modelo, existen parámetros globales que son independientes del modelo. Entre estos parámetros tenemos:

- *repeticiones*: indica el número de veces que deseamos que se ejecute el modelo.

Esta función tiene como objetivo llamar a la función que ejecuta el modelo CNN en este caso *evaluate_model_cnn*, con los parámetros seleccionados por el usuario, tantas veces como repeticiones se haya

indicado. Imprime los resultados de las diferentes ejecuciones, calculando la media de estas ejecuciones y realiza la desviación estándar de estas.

La tercera de las subtareas, de esta épica, es la *adaptación del modelo CNN inicial a un marco de pruebas.* En esta subtarea se permitirá al usuario seleccionar parámetros para entrenar el modelo CNN. El método encargado de realizar esta función es *evaluate_model_cnn* que utiliza como base el modelo CNN inicial creado en el sprint 2.

La estructura que tendrá nuestro marco de pruebas para el modelo CNN es la que se muestra en la *figura 36*. Este modelo seleccionable incluye novedades en la estructura con respeto del modelo creado en el segundo sprint. A diferencia del primer modelo inicial que solo tenía una capa *Conv1d* y una capa *MaxPooling_1D* este modelo permite al usuario seleccionar conjuntos de estas capas. Además, permite seleccionar si se desea *Dropout* y *flatten*. Y por último antes de una capa dense final que dará resultado a nuestro modelo, se permite seleccionar el número deseado de capas dense.

El modelo lo componen las siguientes capas que van en orden y están totalmente conectadas, es decir la salida de una capa es la entrada de la siguiente:

1. *Lista de capas 1D*: Esta es la primera capa de nuestro modelo sequencial CNN, permite seleccionar el número deseado de conjuntos conv1d / MaxPooling1D

2. *Dropout*: Es la segunda posible capa seleccionable de nuestro modelo. El *dropout* es una técnica de regularización para reducir el sobreajuste en las redes neuronales al evitar adaptaciones complejas en los datos de entrenamiento

3. Flatten: El objetivo de esta capa es convertir los datos que recibe en una matriz unidimensional

4. Lista de capas dense: Contiene el número seleccionado por el usuario de capas dense.

5. Dense Final: Es una capa totalmente conectada, es decir recibe información de cada una de las neuronas de la capa previa. Como es la que dará el resultado a nuestro modelo, a esta se le ha dado tamaño 1 y será la última capa de nuestro modelo.

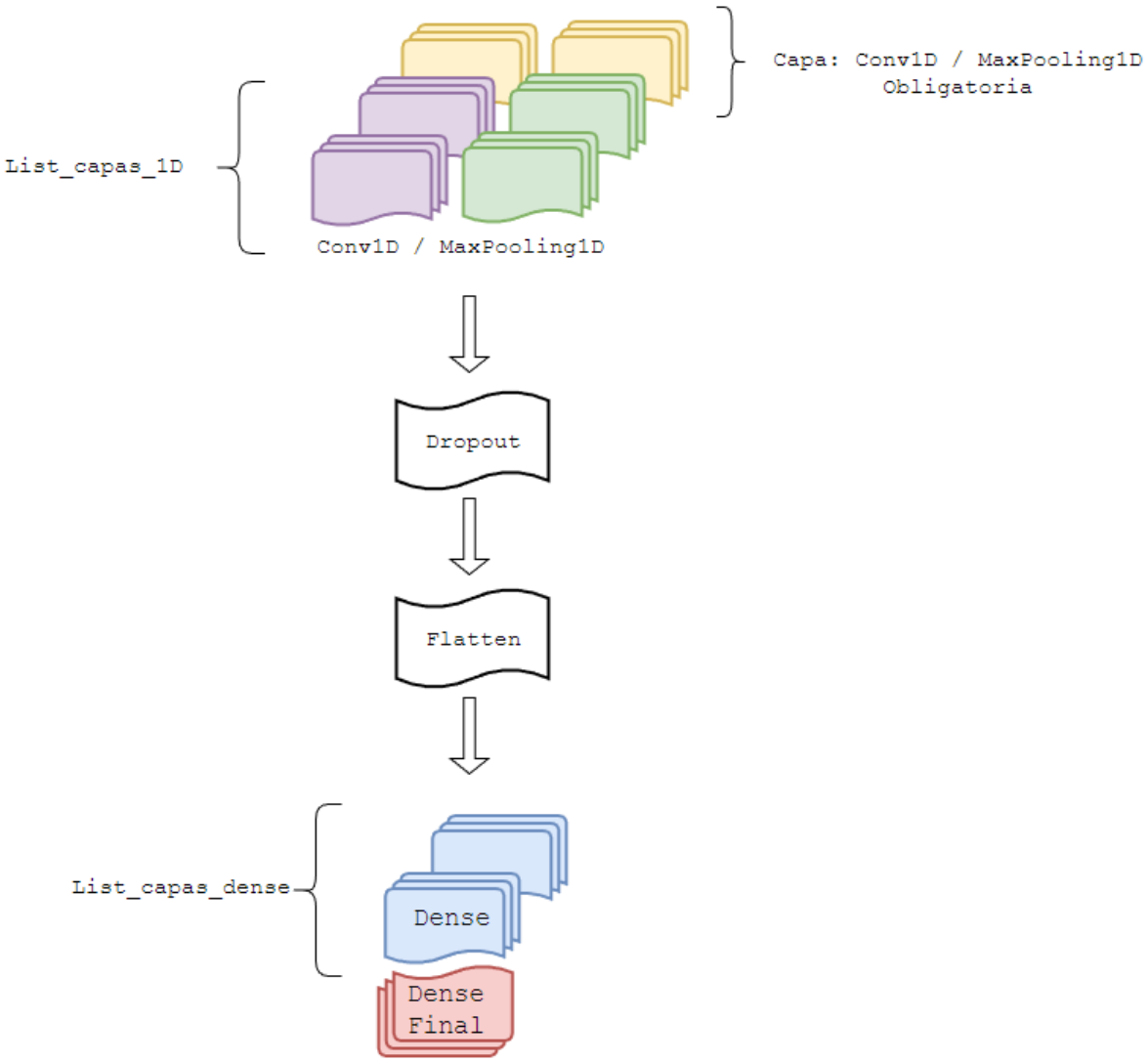


Figura 38 Estructura del marco de pruebas para el modelo CNN

Después de añadir las capas compilamos el modelo. La compilación realizada tiene los parámetros preestablecidos por defecto, entre estos se encuentran la función de perdida '*binary_crossentropy*' como optimizador predeterminado esta '*Adam*' y como métricas deseamos el *accuracy* o acierto.

A continuación, entrenamos el modelo mediante la función *fit*, esta recibe como parámetros los conjuntos de entrenamiento y los conjuntos de prueba. Por último, imprimimos el acierto de la ejecución del modelo y pintamos las gráficas con las funciones mencionadas con anterioridad.

A continuación, se analizan las pruebas de las soluciones implementadas para el desarrollo de las dos épicas de este sprint, *creación de un marco de pruebas para MLP y creación de un marco de pruebas para CNN.*

7.3.4. Pruebas de la solución implementada

En la sección *pruebas y verificación de la solución implementada* se analizarán los resultados de cada una de las implementaciones de las épicas realizadas a lo largo del sprint.

La primera épica, *creación de un marco de pruebas para MLP.* Tiene como objetivo crear un marco de pruebas para un modelo perceptrón multicapa MLP tal como se ha explicado en la sección de implementación de este mismo capítulo. Las pruebas de la solución implementada en esta épica consistirían en la ejecución del modelo MLP creado.

Suponiendo que el usuario ha elegido los siguientes parámetros:

- Hidden_layers: o también llamadas capas ocultas 9
- Max_iterations: 4000
- Repeticiones: 3

```
run_experiment_mlp(data,9,4000,3)
```

A continuación, se muestra el resultado del modelo:

```
<--------------------------- MODELO MLP ---------------------------------------------->

<===========================EJECUCION 1=========================================>
>#1: 68.766

<--------------------------- MODELO MLP ---------------------------------------------->

<===========================EJECUCION 2=========================================>
>#2: 70.866

<--------------------------- MODELO MLP ---------------------------------------------->

<===========================EJECUCION 3=========================================>
>#3: 96.080

<============================ RESULTADOS ========================================>
[68.76640419947506, 70.86614173228347, 96.08048993875765]
Accuracy: 78.571% (+/-12.411)
```

Figura 39 Resultados de ejecución del modelo MLP

Tal como se puede observar en la ejecución, se muestran los resultados de las diferentes ejecuciones y la media de estos con su desviación típica.

La segunda épica, de este sprint es la *creación de un marco de pruebas para CNN* tal como se ha comentado en la sección de implementación de este mismo capítulo. Las pruebas de la solución implementada de esta épica consistirían en la ejecución del modelo CNN creado.

Suponiendo que el usuario ha elegido los siguientes parámetros:

- *Modelo*() contiene:
  - *Capas_1D*: [Capa_1D (256), Capa_1D (128)]
  - *Dropout*: True
  - *Flatten*: True
  - *capas_dense* = [Capa_dense(64),Capa_dense(),Capa_dense(12)]

- *Options*() contiene:
  - *n_kernel*: 5
  - *epoch*: 20
  - *batch_size*: 64

- *Repeticiones*: 1

```
run_experiment_cnn(modelo1,data,options,1)
```

A continuación, se muestra la ejecución del modelo generado, en primer lugar, se imprime la estructura de capas del modelo tal como se muestra en la *figura 38*

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Conv_1 (Conv1D)             (None, 8, 64)             384

 MaxPool_1 (MaxPooling1D)    (None, 4, 64)             0

 Additional_Conv_53 (Conv1D)  (None, 4, 256)           82176

 Additional_MaxPool_22 (MaxP  (None, 2, 256)           0
 ooling1D)

 Additional_Conv_39 (Conv1D)  (None, 2, 128)           163968

 Additional_MaxPool_28 (MaxP  (None, 1, 128)           0
 ooling1D)

 Dropout (Dropout)           (None, 1, 128)            0

 Flatten (Flatten)           (None, 128)               0

 Dense_66 (Dense)            (None, 64)                8256

 Dense_70 (Dense)            (None, 128)               8320

 Dense_92 (Dense)            (None, 12)                1548

 Dense_Last (Dense)          (None, 1)                 13

=================================================================
Total params: 264,665
Trainable params: 264,665
Non-trainable params: 0
_________________________________________________________________
```

Figura 40 Estructura de capas de la ejecucion del marco de pruebas del modelo CNN

A continuacion se imprime la grafica que genera la funcion *pintar_grafica_epoch*, en la que se hace una comparacion de la precision de los diferentes conjuntos de datos entrenamiento y prueba y su evolucion a largo de los epoch.

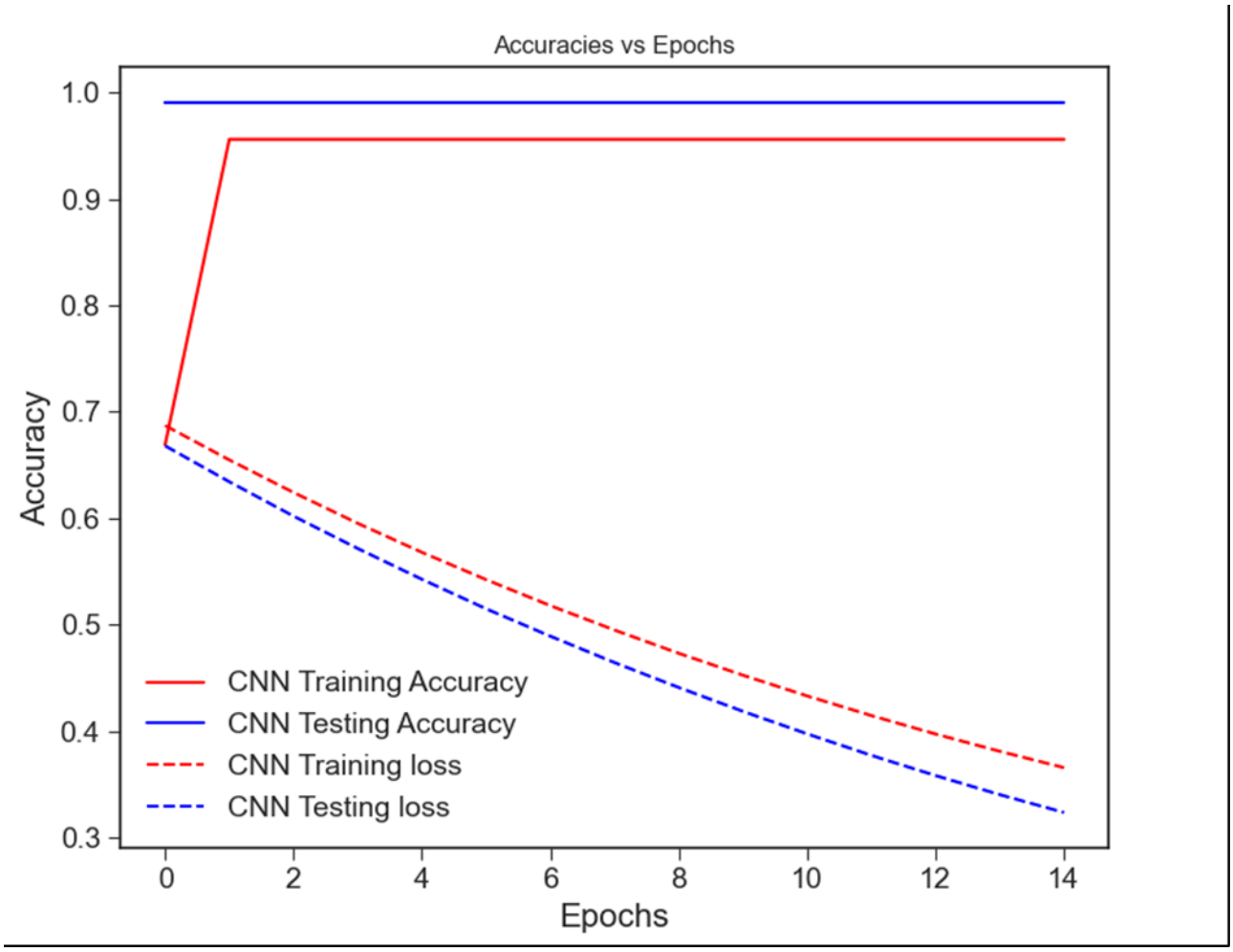


Figura 41 Grafica de acierto versus Epoch

En esta última figura se muestra la matriz de confusión y los resultados de ejecución del modelo. La matriz de confusión se calcula de forma global, una vez fijados los parámetros y ejecutado el modelo. En este proyecto no nos concentraremos en los resultados que ofrece esta, es un experimento de prueba.

```
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98      1086
         1.0       0.00      0.00      0.00        50

    accuracy                           0.96      1136
   macro avg       0.48      0.50      0.49      1136
weighted avg       0.91      0.96      0.93      1136

Confusion Matrix :
 [[1086    0]
 [  50    0]]
Accuracy :  0.9559859154929577
Sensitivity :  1.0
Specificity :  0.0
False PR :  0.0
False NR :  1.0

<============================EJECUCION 1=========================================>
>#1: 99.108

<========================== RESULTADOS ==========================================>
[99.1076111793518]
Accuracy: 99.108% (+/-0.000)
```

Figura 42 Análisis del modelo CNN

7.4. SPRINT 4: Interfaz Gráfica

Para el desarrollo de esta sección se ha migrado el código de Google Colab de los Sprint anteriores al IDE pytorch. Como patrón utilizado para el desarrollo de la interfaz destacamos el Modelo-Vista-Controlador.

7.4.1. Objetivos

Una vez creada la plataforma de pruebas, lo que nos queda por desarrollar es la interfaz gráfica del proyecto. El objetivo de este Sprint es crear una interfaz gráfica en Python que permitirá al usuario interactuar con los marcos de pruebas creados.

7.4.2. Análisis de objetivos

Los objetivos en Scrum, son metas tangibles redactadas en conjunto por el equipo que están sujetas a la duración del sprint. Por tanto, el objetivo de este sprint al igual que en los anteriores es completar las historias de usuario o épicas que se detallan a continuación:

- Sprint 4: Interfaz gráfica
    - Épica 1: Creación de la interfaz gráfica
    - Épica 2: Documentación

Como se puede observar de la descripción anterior, **como primera épica** de este sprint, tenemos la *Creación de la interfaz gráfica*, esta permite al usuario interactuar con el sistema. Entre las acciones que el usuario puede realizar, se cuenta el procesamiento de datos en crudo, el entrenamiento de los modelos MLP y CNN y la lectura de un FAQ o preguntas frecuentes entre otras.

Al ser la única tarea de nuestro Sprint es la más laboriosa y se compone de varias subtareas que se detallan a continuación:

- Creación de un menú principal.
- Creación de la vista de procesamiento de datos.
- Creación de la vista de entrenamiento de modelos.
- Creación de la vista FAQ.

Por último, como *tercera épica*, al igual que sucede en cada sprint tenemos como historia la documentación. Esta tarea consiste en redactar y documentar lo realizado a lo largo de este Sprint.

A continuación, se analiza cómo se ha realizado el desarrollo de la única épica de este sprint Creación de la interfaz gráfica.

7.4.3. Desarrollo e implementación

La **primera subtarea** de nuestra épica, es la *Creación de un menú principal*. El menú principal tiene como objetivo permitir al usuario acceder a los recursos importantes del sistema. En nuestro caso el menú principal permite acceder a las siguientes vistas:

- Vista de procesamiento de datos
- Vista de entrenamiento de modelos
- Vista de FAQ

Tal como se ha mencionado en el Marco de trabajo de este proyecto para el desarrollo de la interfaz se ha pasado de utilizar Google colab a utilizar el IDE pycharm. Este tiene mayor capacidad de procesamiento que otros editores de código como es Visual Studio Code y permite la creación de interfaces.

La vista del menú principal se compone de tres botones:

- Data Preprocessing: Botón que nos dirige a la vista de procesamiento de datos
- Model Training: Botón que nos dirige a la vista de entrenamiento de modelos
- FAQs: Botón que nos dirige a una sección de preguntas frecuentes

La **segunda subtarea** de nuestra épica, es la *Creación de la vista de procesamiento de datos*. La vista de procesamiento de datos tiene como objetivo permitir al usuario procesar los datos en crudo y guardar los datos una vez procesados.

La vista del menú principal se compone de 4 botones:

- Select Data From Device: Botón que nos permite seleccionar los datos en local que deseamos procesar y guarda el fichero una vez procesado
- Select Data From G-Drive: Botón que nos permite hacer login en Google Drive y descargar los datos en crudo desde la red, para después seleccionarlos desde local.
- Main Menu: Botón que nos lleva a la vista del menú principal
- Training FAQs: Botón que nos dirige a una sección de preguntas frecuentes

La **tercera subtarea** de nuestra épica, es la *Creación de la vista de entrenamiento de modelos*. La vista de entrenamiento de modelos tiene como objetivo permitir al usuario seleccionar los parámetros de los modelos MLP y CNN, entrenar estos modelos y obtener los resultados de estos como se ha mostrado en el sprint anterior.

La vista de entrenamiento de modelos ha sido una de las más complicadas de realizar, debido a la cantidad de parámetros del modelo CNN.

La vista de entrenamiento se compone de 5 botones principales:

- Train Model: Es el botón que ejecuta el modelo, ya sea CNN o MLP, solo se puede pulsar si están seleccionados todos los parámetros del modelo seleccionado.
- Load Processed Data: Botón que nos permite cargar el csv de datos ya procesados.
- Reset Values: Botón que nos permite resetear los datos seleccionados
- Export Anysis: Botón que nos permite guardar el resultado del modelo y el análisis

La **cuarta subtarea** de nuestra épica, es la *Creación de la vista FAQ*. La vista de FAQ tienen como objetivo ofrecer información general sobre el proyecto y su funcionalidad. En esta vista también se han añadido los datos de contacto del desarrollador, en la que se pueden indicar cambios o mejoras en el proyecto.

7.4.4. Pruebas de la solución implementada

Para esta sección se ha creado un nuevo capítulo en el que se mostraran las diferentes vistas de nuestro proyecto y las interacciones que podemos tener.

Capítulo 8
Plataforma de trabajo

En este capítulo se mostrará el diseño de la interfaz de este proyecto y las funcionalidades que ofrece esta. Para mostrar la plataforma de trabajo utilizaremos de guía las subtareas del *sprint 4 Interfaz gráfica* que son las siguientes:

- Creación de un menú principal.
- Creación de la vista de procesamiento de datos.
- Creación de la vista de entrenamiento de modelos.
- Creación de la vista FAQ.

La implementación de **la primera subtarea**, creación de un menú principal es la siguiente:

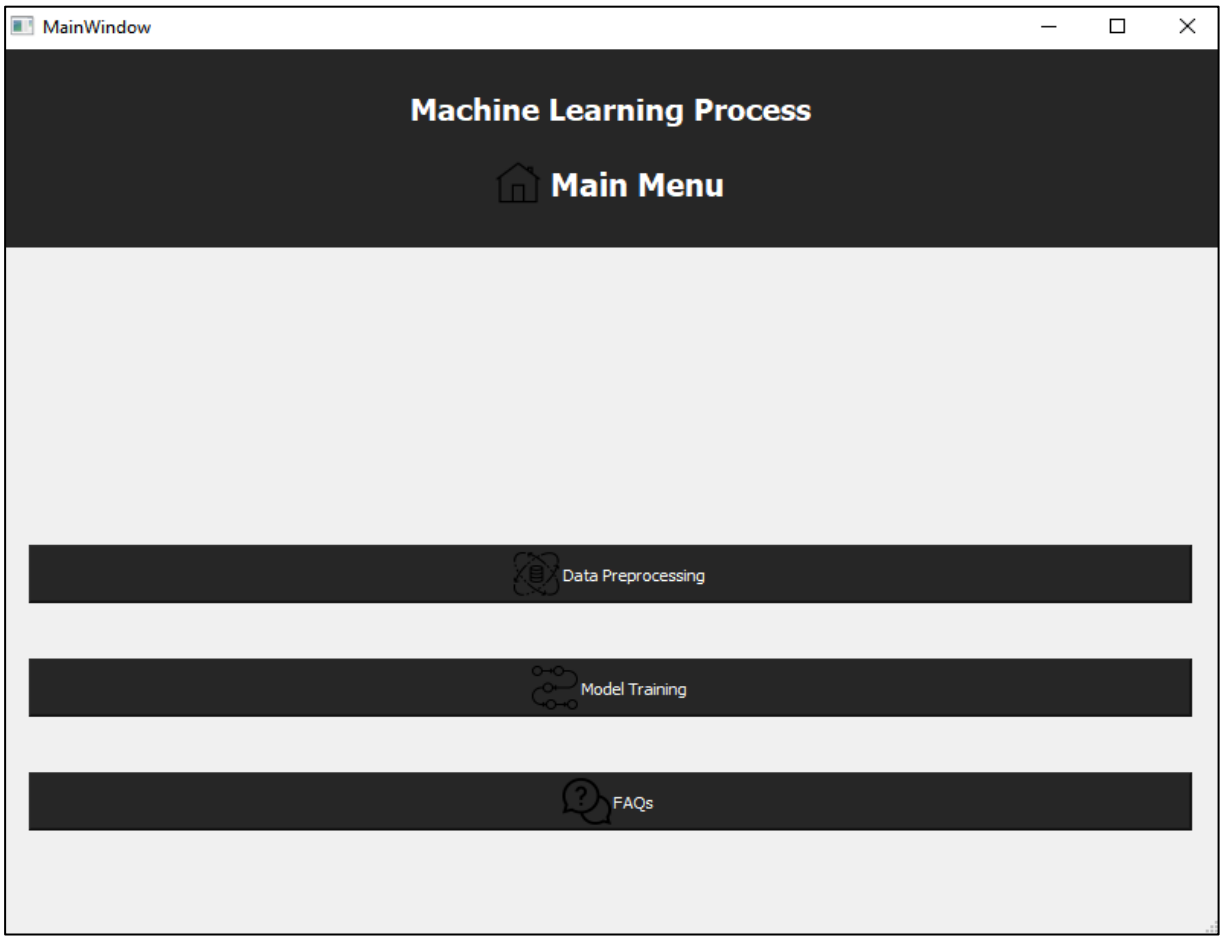


Figura 43 Menú principal de la aplicación

Tal como se puede observar contiene los tres botones mencionados en la seccion de implementacion, Data Preprocessing, Model Training y FAQs

Al iniciar la apliacion aparece el menu principal y un log donde se mostraran las ejecuciones y otra informacion de interes del sistema.

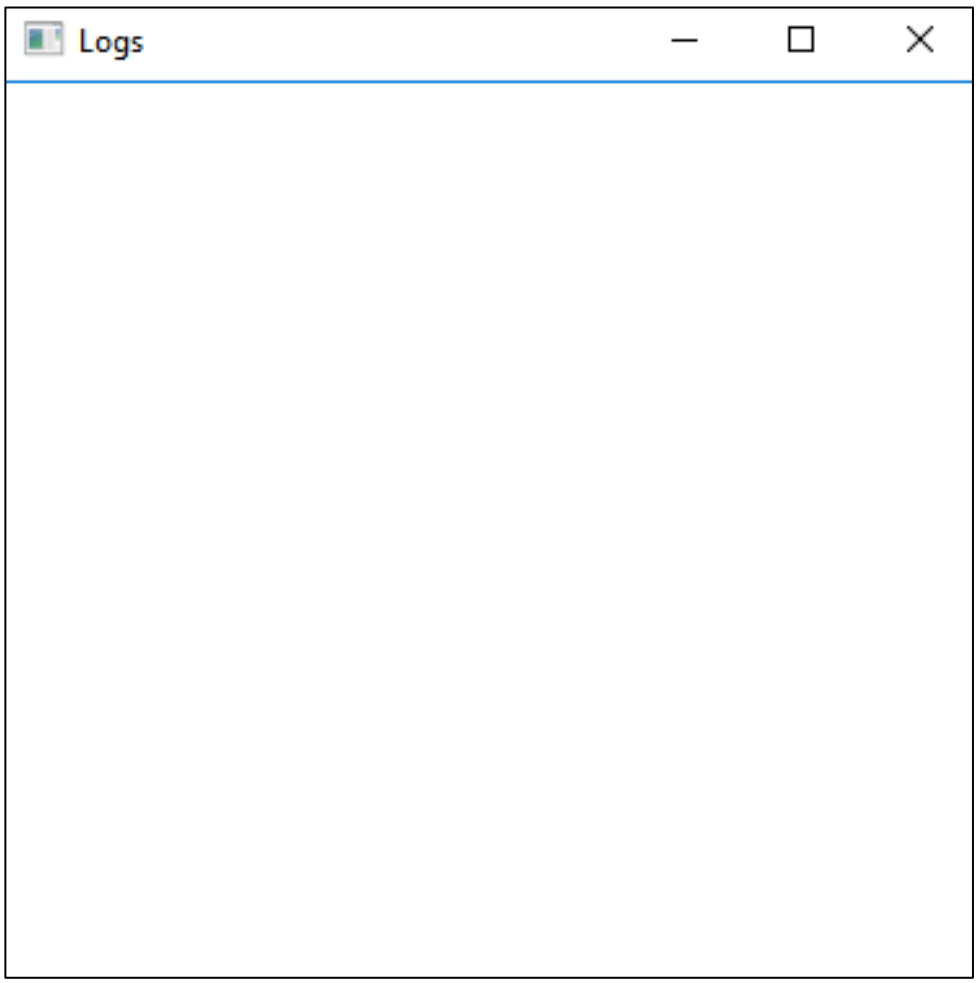


Figura 44 Ventana de Logs de la aplicación

Este log empieza estando en blanco y a medida que interactuamos con el sistema se va rellenando

La implementación de **la segunda subtarea**, creación de la vista de procesamiento de datos es la siguiente:

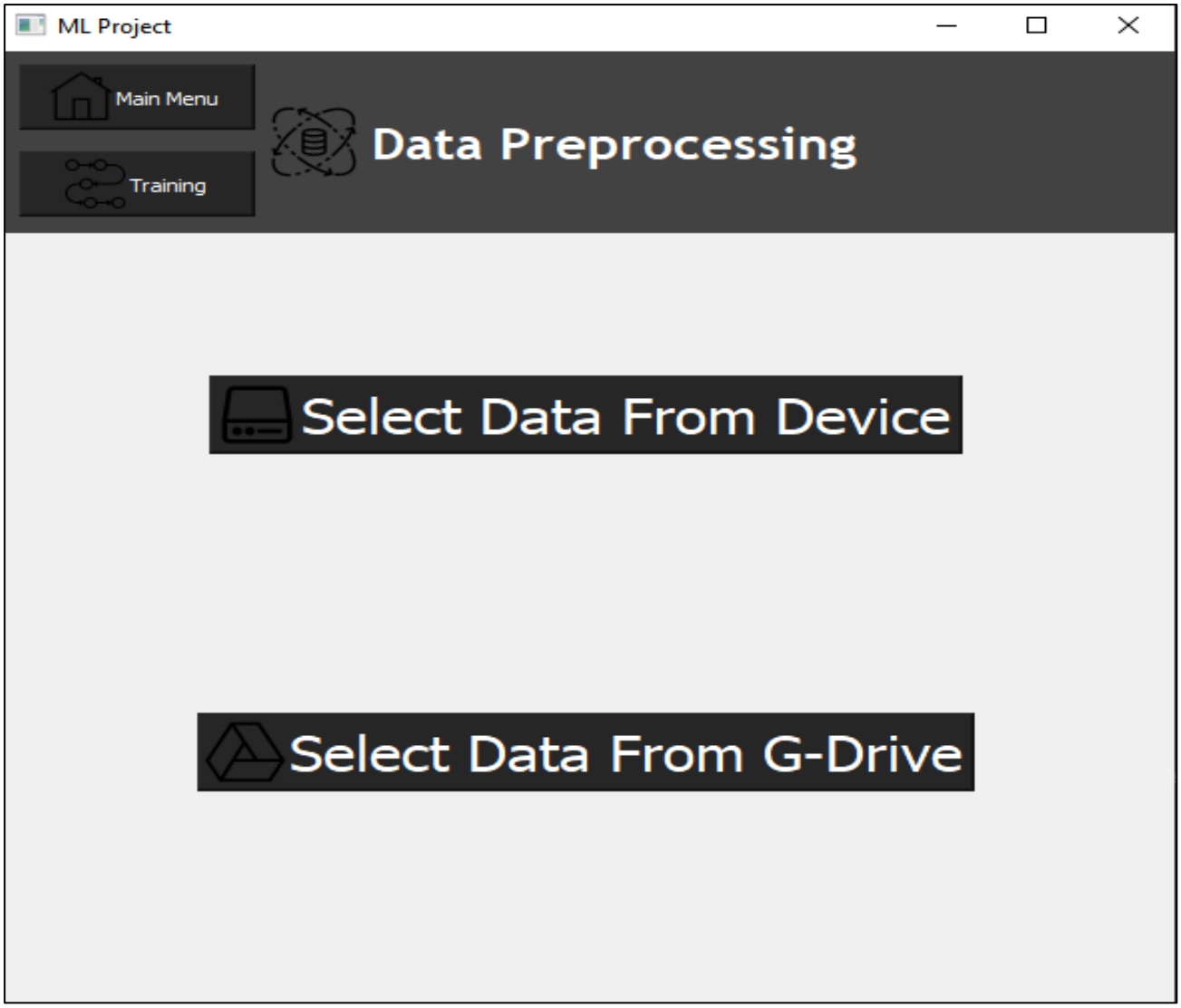


Figura 45 Ventana de preprocesado de datos

Si pulsamos el primer botón, Select Data From Device tendremos la siguiente secuencia:

1. Seleccionar la carpeta que contiene el conjunto TFG
2. Seleccionar la carpeta que contiene el conjunto TFM

3. Seleccionamos donde deseamos guardar el fichero

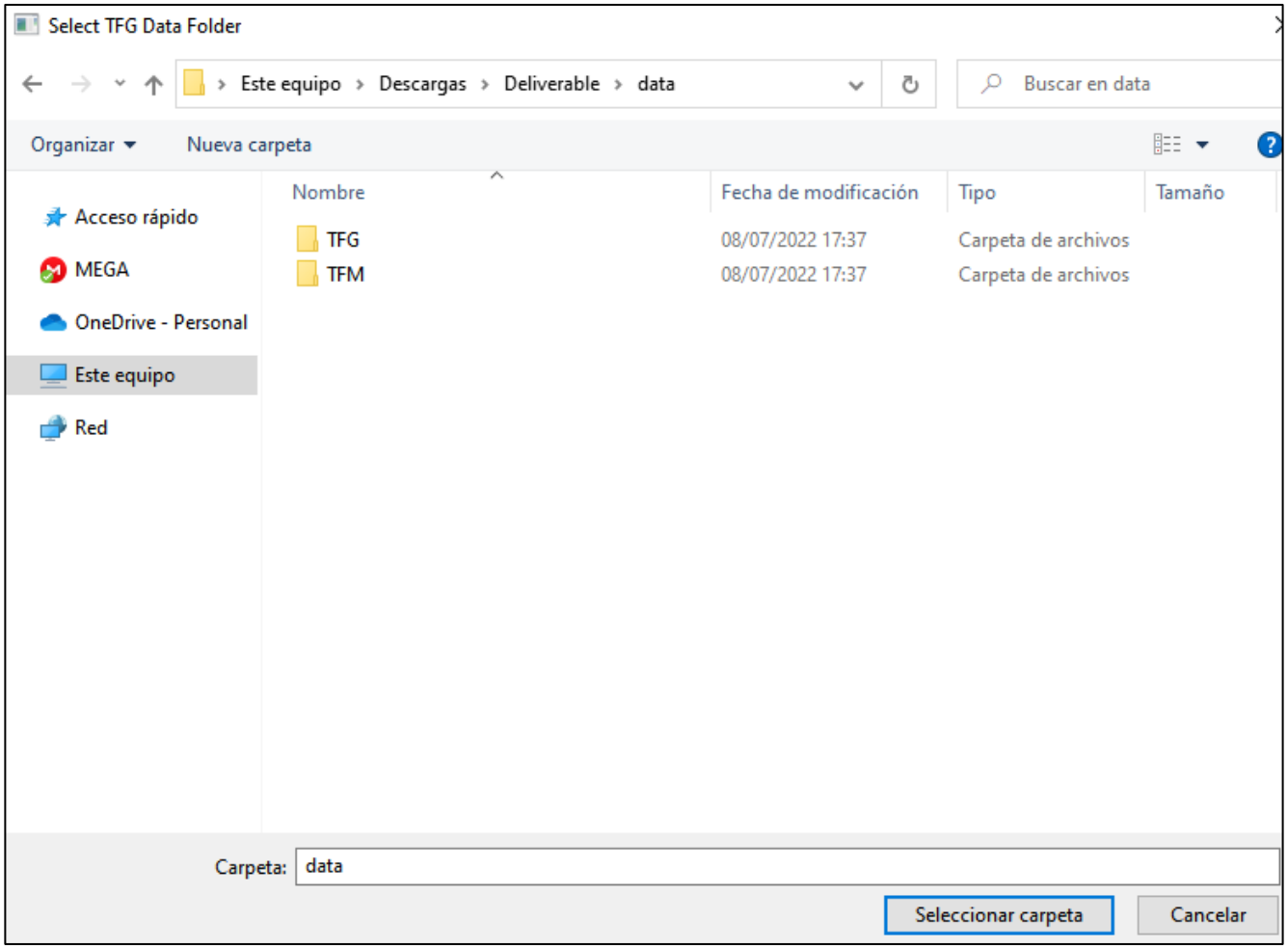


Figura 46 Cuadro de selección de conjunto de datos TFG

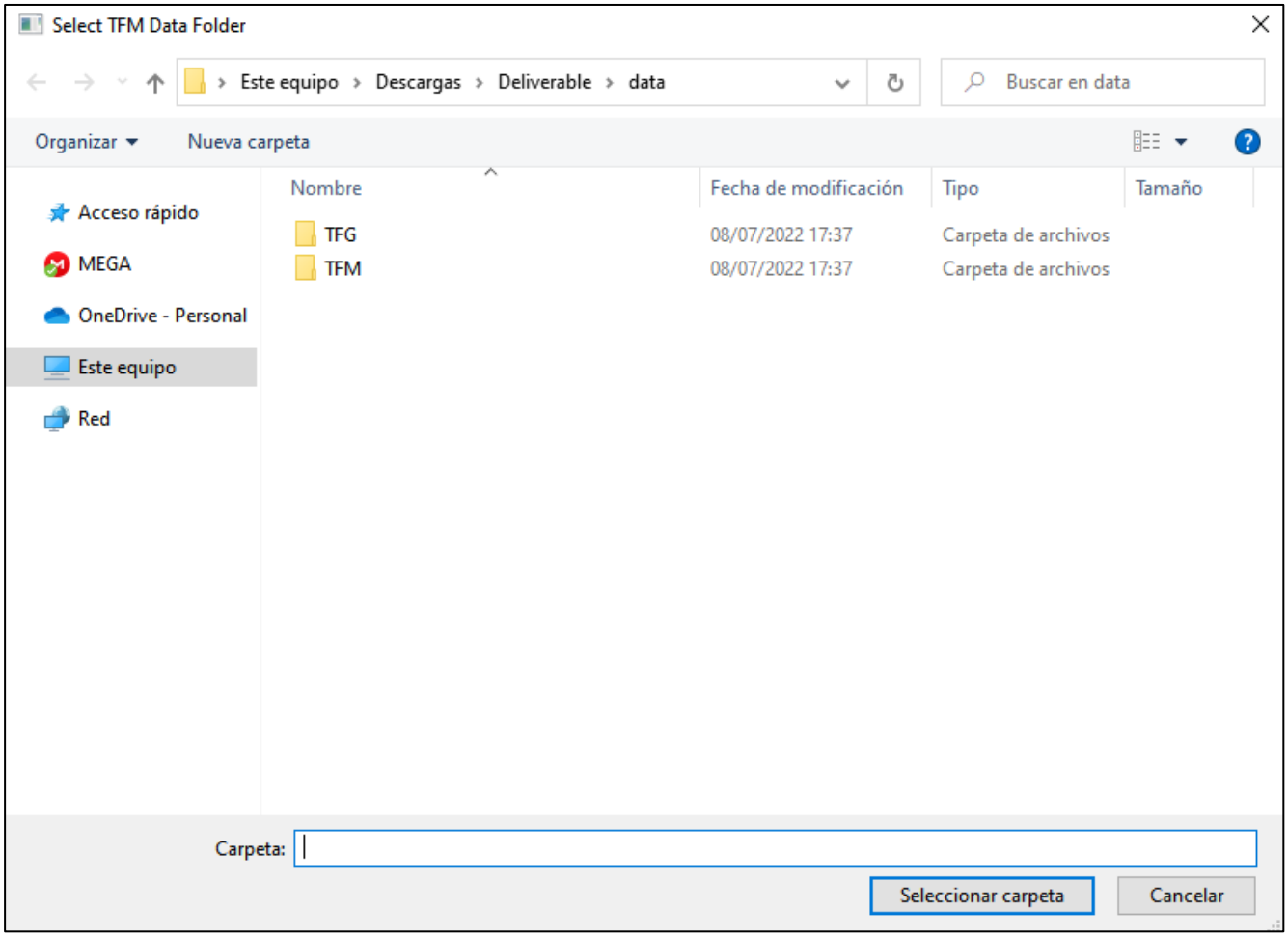


Figura 47 Cuadro de selección de conjunto de datos TFM

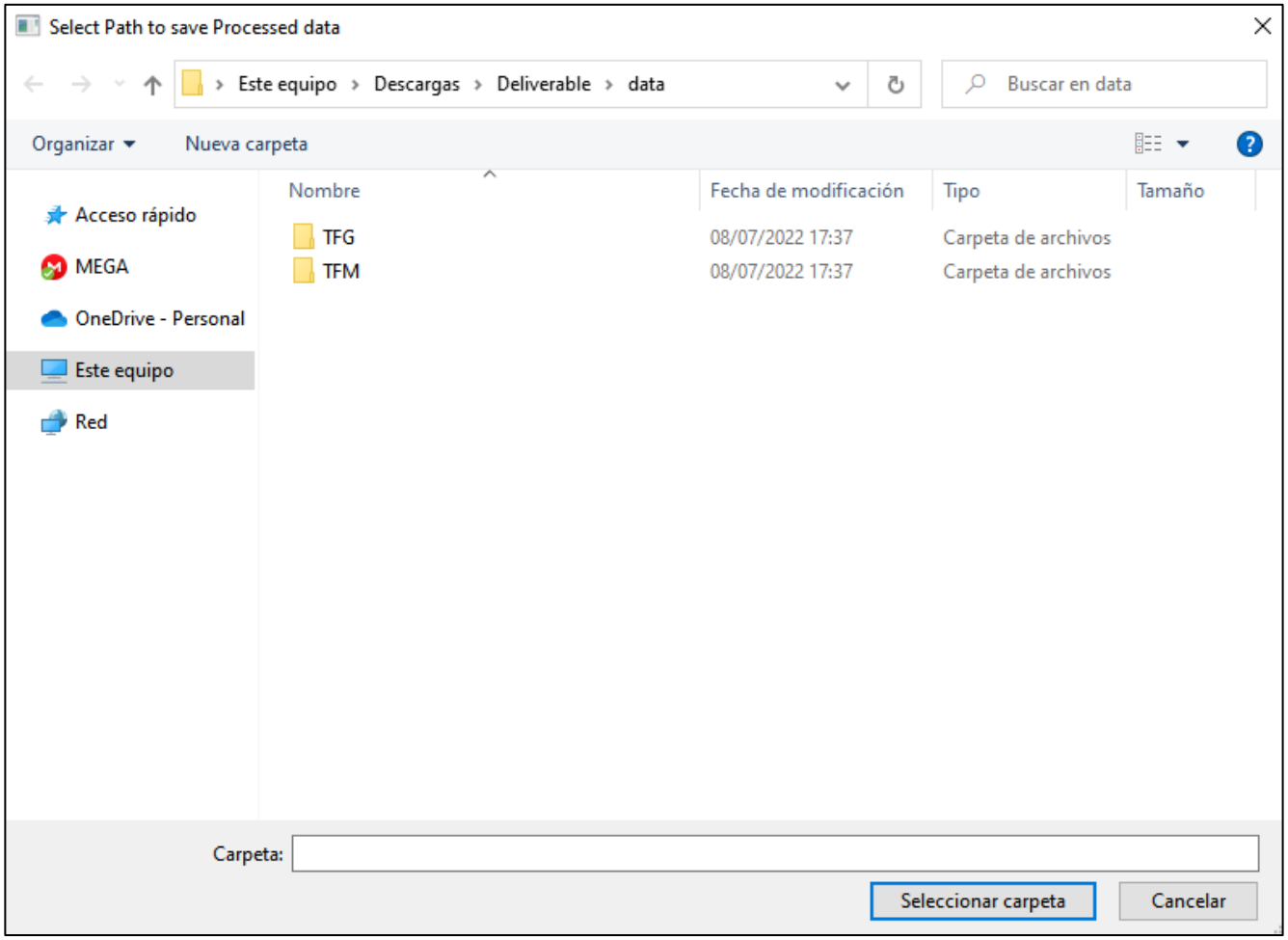


Figura 48 Cuadro de selección de directorio para el almacenamiento del fichero procesado

Una vez realizada la secuencia anterior, nos aparecerá una ventana de cargando hasta que finalice el proceso, que tarda alrededor de 25 minutos.

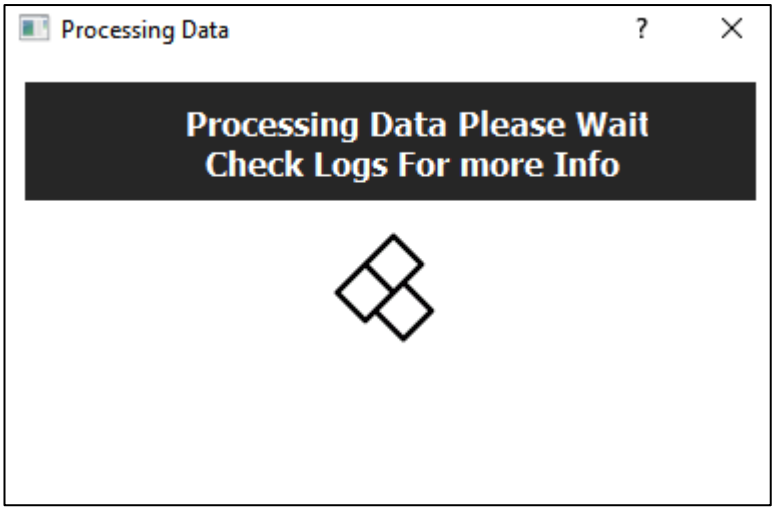


Figura 49 Ventana de loading

Si seleccionamos Select Data From G-Drive, iniciaremos un proceso de autenticación en Google Drive para descargar los ficheros ponibles desde la red.

Una vez que nos autentiquemos recibiremos este mensaje de la página web:

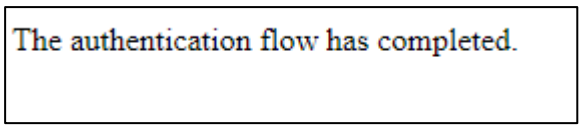


Figura 50 Mensaje de autenticación de Drive completada con éxito

Y en la ventana de procesamiento de datos aparecerán dos nuevos campos

- TFM: Ruta Drive de datos TFM
- TFG: Ruta Drive de datos TFG

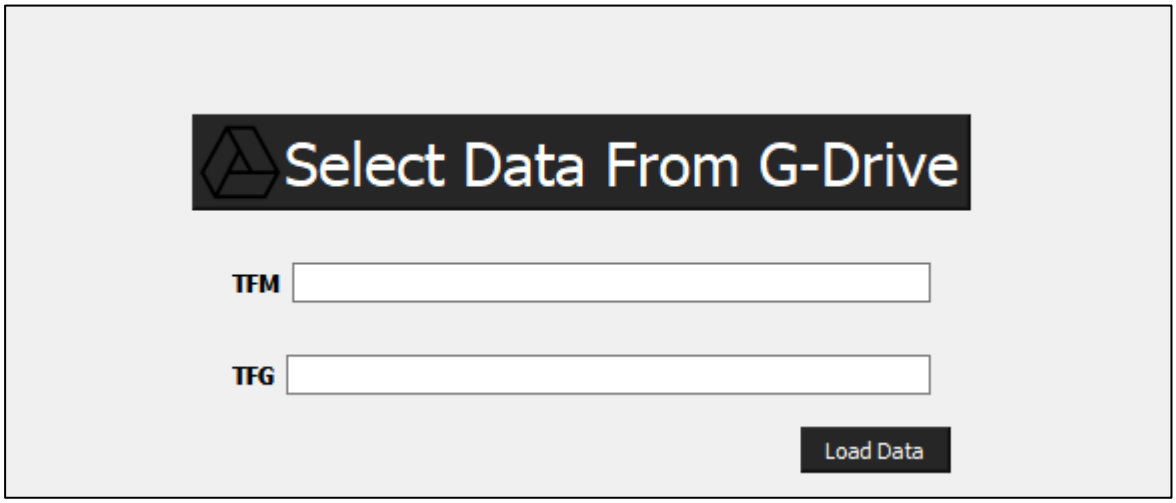


Figura 51 Nuevos datos a introducir en la seccion Select Data From G-Drive

A continuación, se muestra como obtener la dirección del conjunto TFG.

1. Ir al directorio que contiene el conjunto TFG en drive
2. Copiar el path seleccionado, como se muestra en la siguiente figura

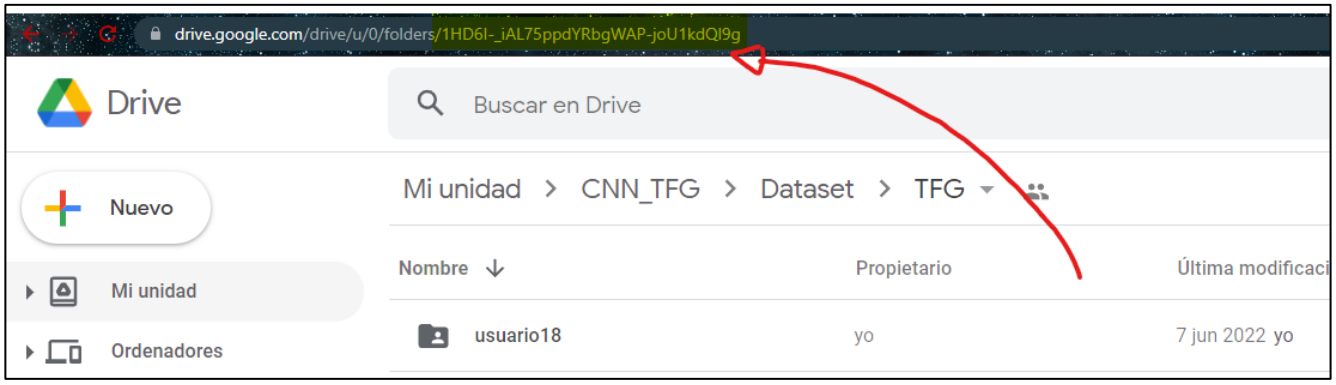


Figura 52 Path que debemos copiar para el conjunto TFG

Una vez realizado esto con ambos conjuntos pulsamos Load Data:

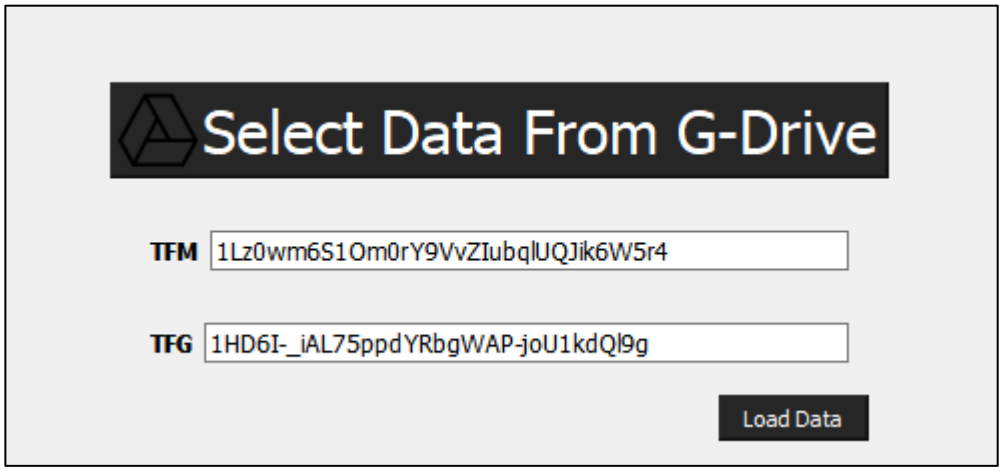


Figura 53 Datos de path para ambos conjuntos

Y seleccionamos la carpeta donde queremos que se descarguen ambos conjuntos. En el ejemplo mostrado la carpeta será descarga_drive.

El proceso de descarga empieza, y se va mostrando el proceso en el log.

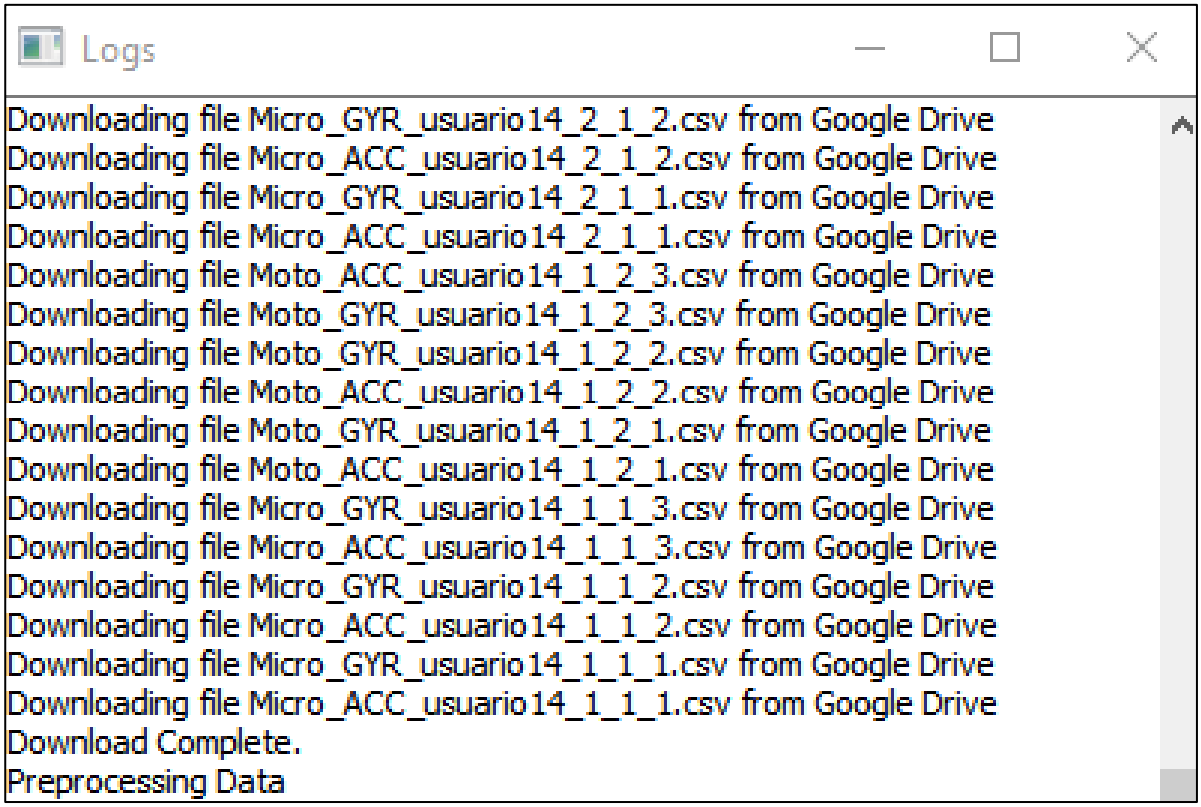


Figura 54 Descarga de ficheros y directorios a partir de G-Drive

Finalizada el proceso, obtendremos ambos conjuntos de datos guardados en la carpeta seleccionada:

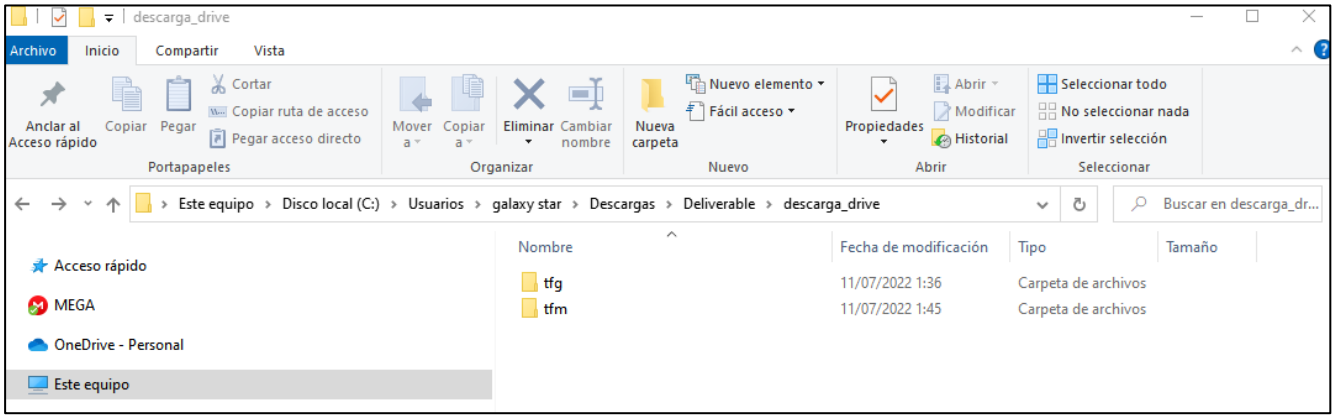


Figura 55 Descarga de conjunto de directorios completada

Y se nos abre un promt pidendo que seleccionemos la carpeta donde se han procesado los datos, en nuestro caso descarga_drive.

Al seleccionarla, se procesan los datos tal como se procesaran en local. Nos aparecerá una ventana de cargando hasta que finalice el proceso, que tarda alrededor de 25 minutos

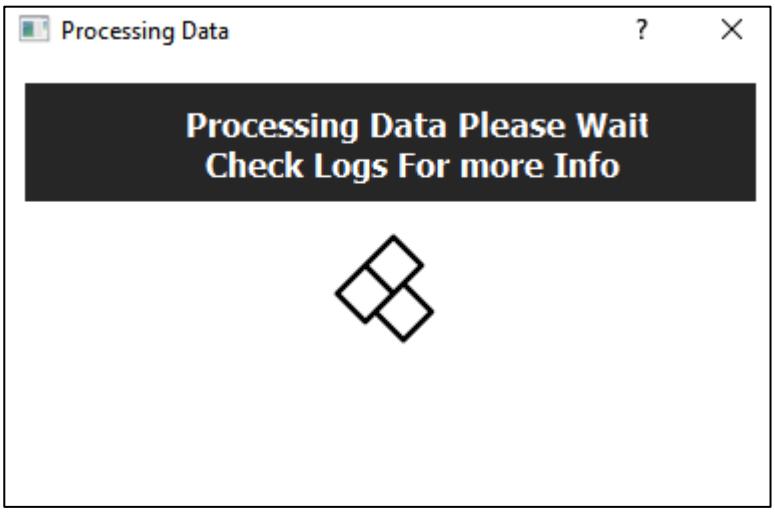


Figura 56 Ventana de espera de procesamiento de datos

La implementación de **la tercera subtarea**, creación de la vista de entrenamiento es la siguiente:

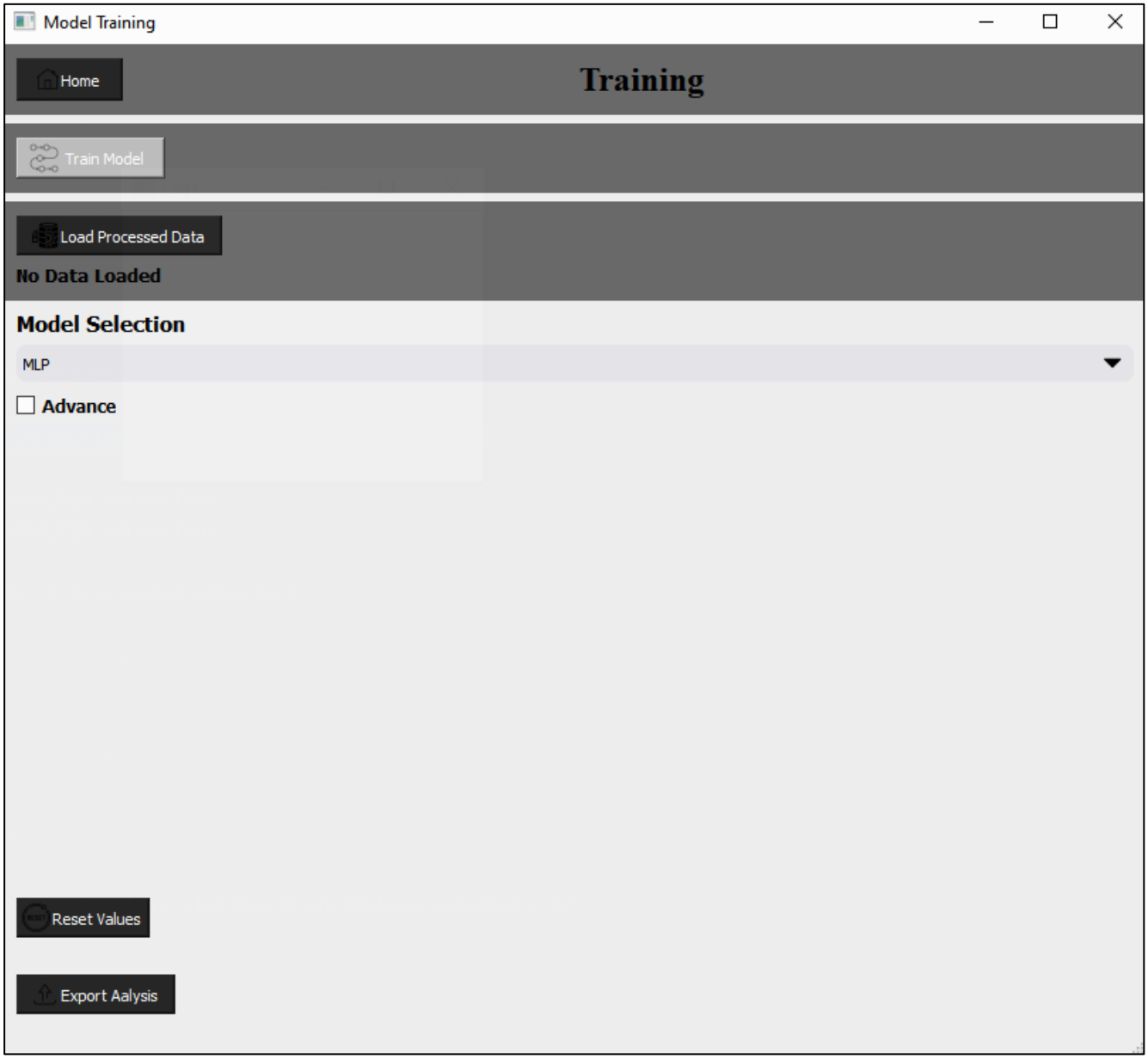


Figura 57 Ventana de Entrenamiento de la plataforma

Lo primero que debemos hacer es seleccionar Load processed Data y se nos abrirá una ventana en la que seleccionamos el fichero de csv procesado

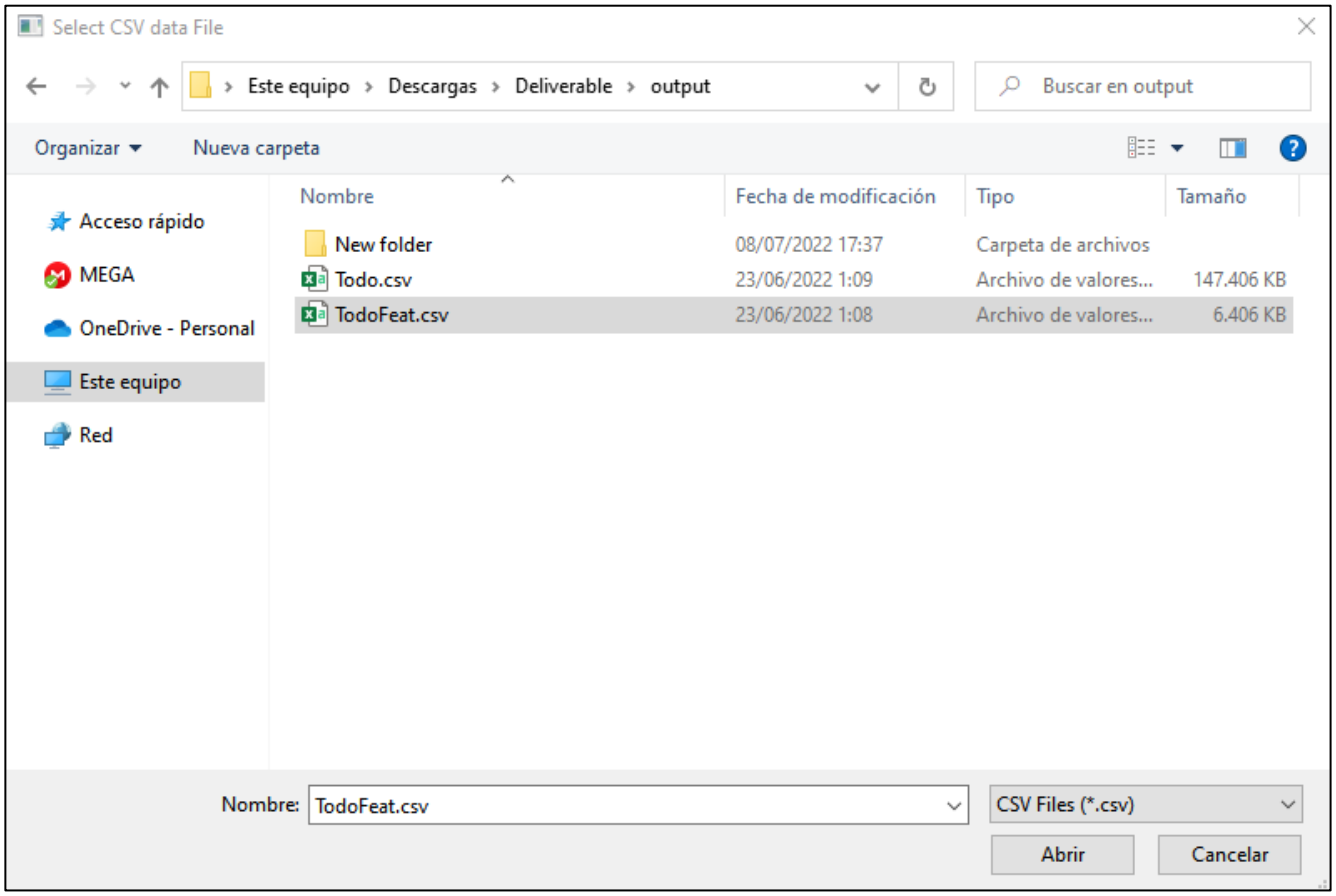


Figura 58 Cuadro de dialogo que se abre para seleccionar el fichero procesado

Una vez seleccionados el csv que contiene los datos nos aparecerá el path de este fichero debajo del botón Load Processed Data.

Figura 59 Confirmación de la aplicación de que el fichero se ha cargado con éxito

Ahora tenemos dos opciones, seleccionar el modelo cnn o el modelo mlp. Si deseamos elegir nosotros los parámetros seleccionamos la opción Advance y se desplegaran las opciones:

El modelo CNN tiene las siguientes opciones avanzadas:

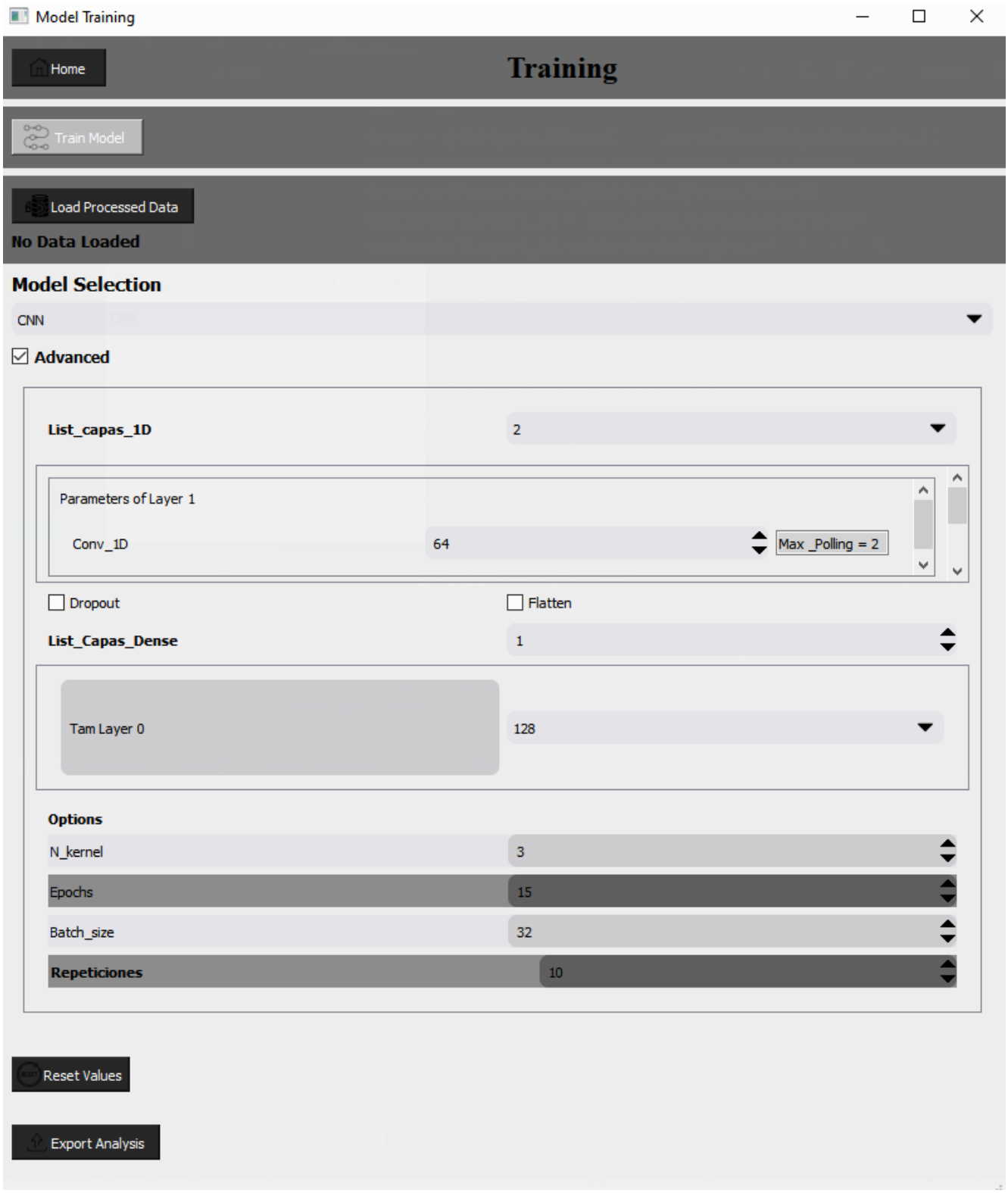


Figura 60 Vista de los parámetros seleccionables del modelo CNN

El modelo MLP tiene las siguientes opciones avanzadas:

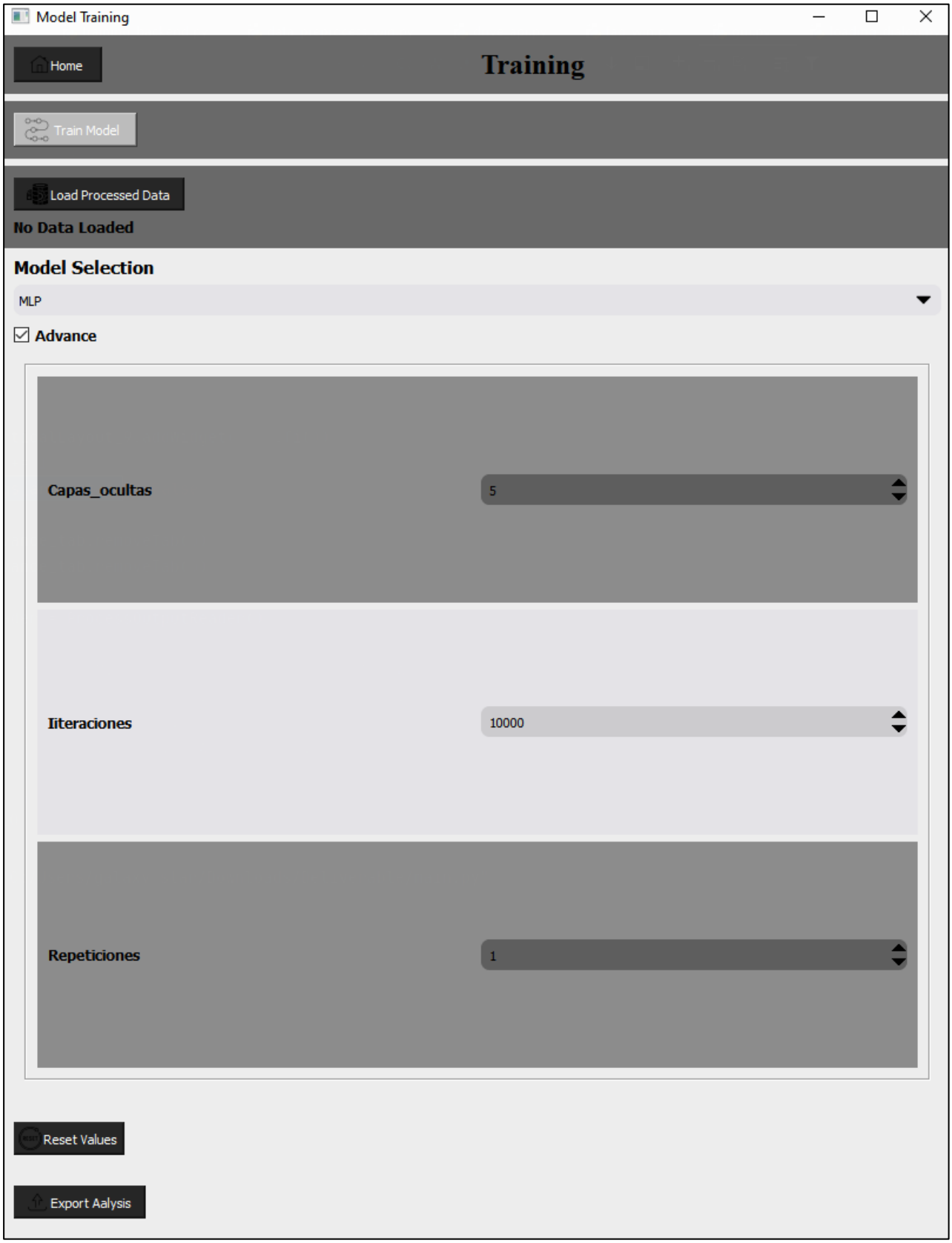


Figura 61 Vista de los parámetros seleccionables del modelo MLP

Una vez seleccionados los parámetros y cargados los datos, el botón train model se puede pulsar y el modelo se entrenará.

La ejecución del MLP si no elegimos opciones avanzadas es el siguiente:

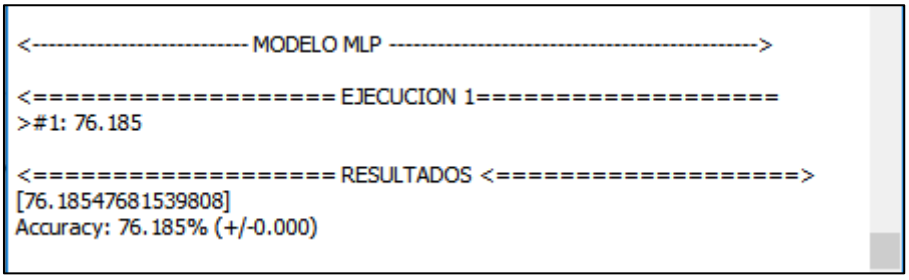


Figura 62 Resultado de ejecucion del modelo CNN

Podemos exportar los resultados pulsando el botón Export Analysis y nos pedirá la carpeta donde deseamos exportar el análisis.

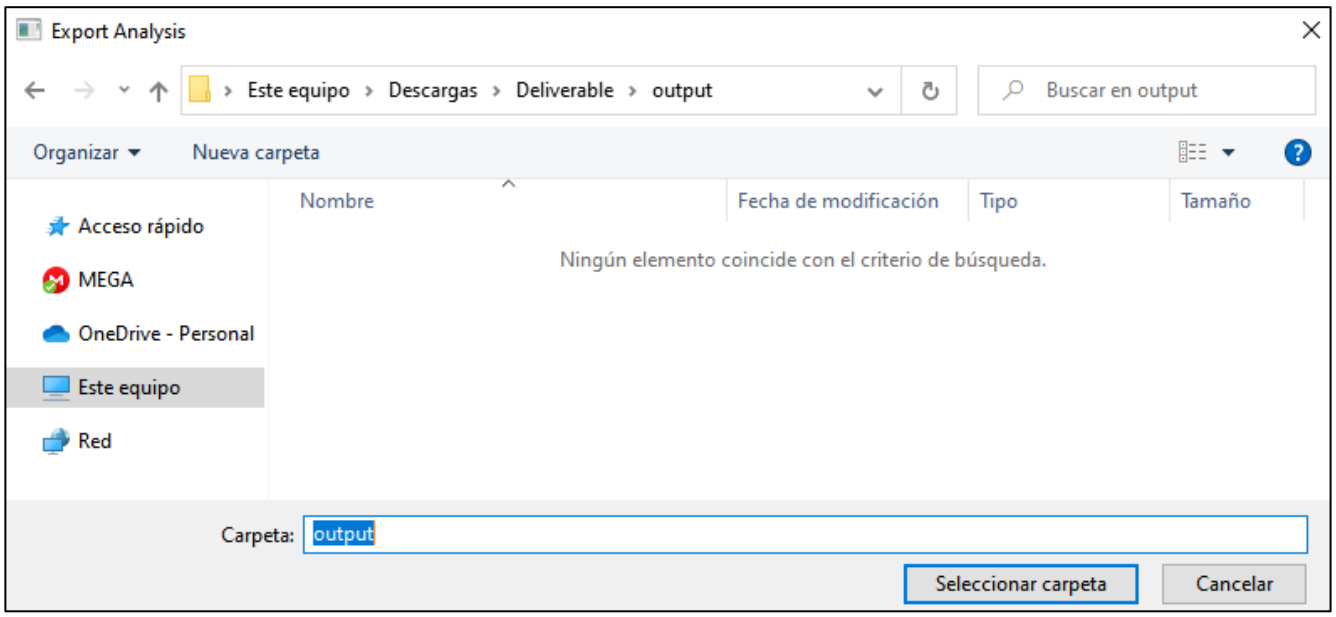


Figura 63 Cuadro de dialogo de exportar analisis

Se confirma que el análisis se ha guardado

Figura 64 Ruta de almacenamiento del análisis

La ejecución del modelo CNN si no elegimos opciones avanzadas es el siguiente:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Conv_1 (Conv1D)              (None, 10, 64)            256

MaxPool_1 (MaxPooling1D)     (None, 5, 64)             0

Additional_Conv_73 (Conv1D)  (None, 5, 64)             12352

Additional_MaxPool_71 (MaxP  (None, 2, 64)             0
ooling1D)

Additional_Conv_12 (Conv1D)  (None, 2, 128)            24704

Additional_MaxPool_36 (MaxP  (None, 1, 128)            0
ooling1D)

Dropout (Dropout)            (None, 1, 128)            0

Flatten (Flatten)            (None, 128)               0

Dense_78 (Dense)             (None, 64)                8256

Dense_62 (Dense)             (None, 128)               8320

Dense_72 (Dense)             (None, 12)                1548

Dense_Last (Dense)           (None, 1)                 13

=================================================================
Total params: 55,449
Trainable params: 55,449
Non-trainable params: 0
```

Figura 65 Estructura de capas del modelo CNN

```
  _warn_prf(average, modifier, msg_start, len(result))
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98      1086
         1.0       0.00      0.00      0.00        50

    accuracy                           0.96      1136
   macro avg       0.48      0.50      0.49      1136
weighted avg       0.91      0.96      0.93      1136

Confusion Matrix :
 [[1086    0]
 [  50    0]]
Accuracy :  0.9559859154929577
Sensitivity :  1.0
Specificity :  0.0
False PR :  0.0
False NR :  1.0

<==================== EJECUCION 1<====================>
>#1: 99.108

<==================== RESULTADOS <====================>
[99.1076111793518]
Accuracy: 99.108% (+/-0.000)
```

Figura 66 Resultados del análisis de la ejecución del modelo CNN

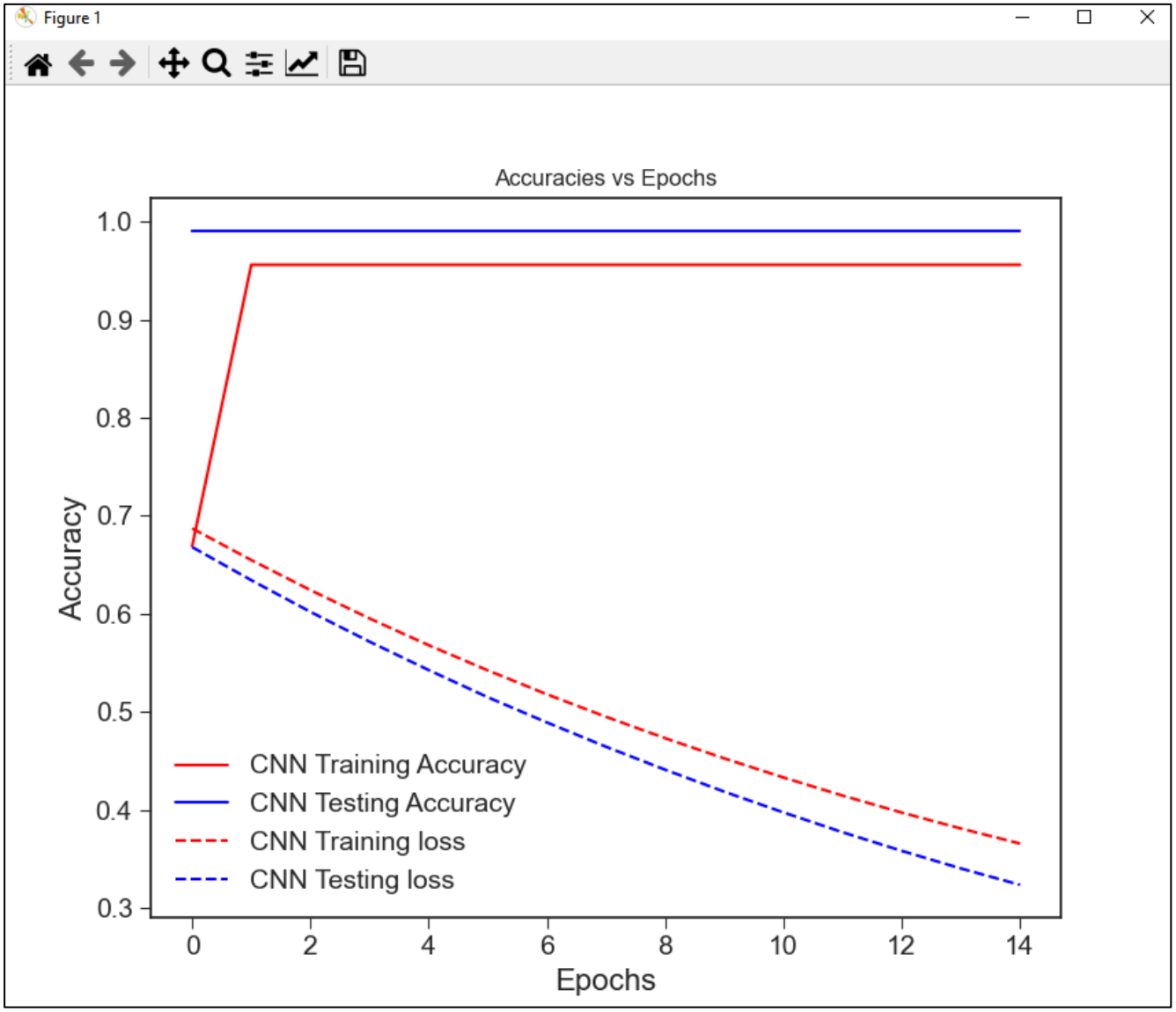


Figura 67 Grafica resultante de la ejecución del modelo

Si deseamos guardar el grafico pulsamos el botón de guardar en la figura y seleccionamos la ubicación donde deseamos guardar el análisis.

Podemos exportar los resultados pulsando el botón Export Analysis y nos pedirá la carpeta donde deseamos exportar el análisis.

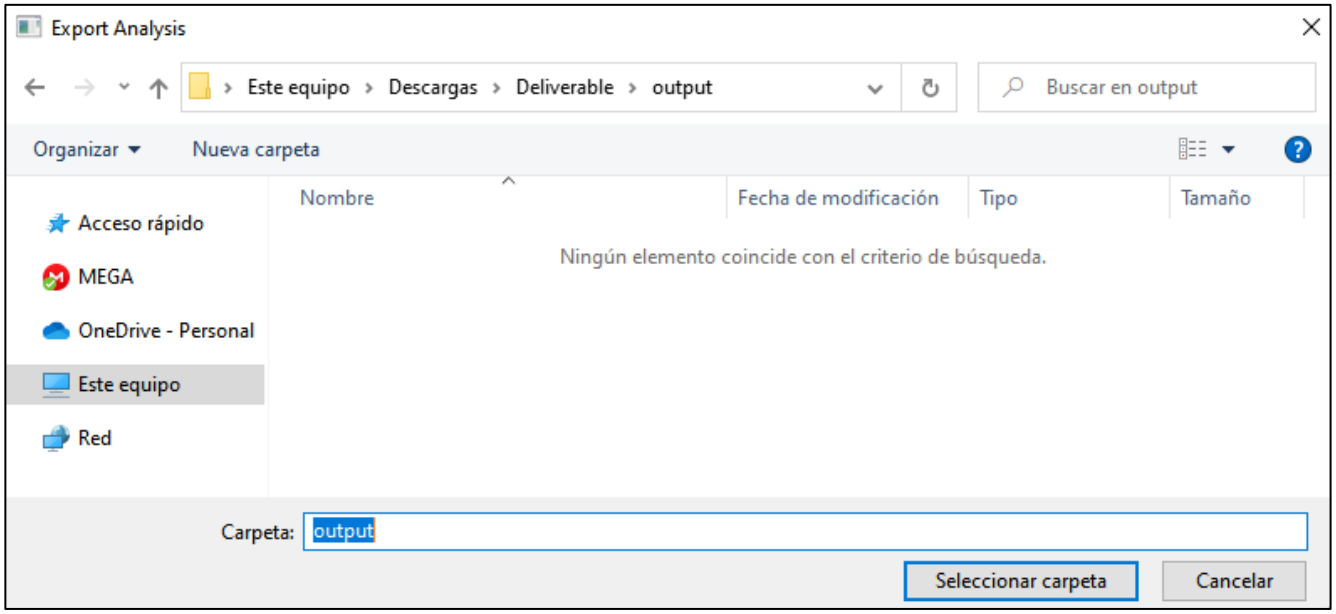


Figura 68 Cuadro de diálogo para el almacenamiento del análisis

Se confirma que el análisis se ha guardado

Figura 69 Ruta de almacenamiento del análisis

La implementación de **la 4 subtarea**, creación de la vista de FAQ es la siguiente:
Es una vista que no es interactiva y menciona conceptos del Machine Learning y de las Redes CNN

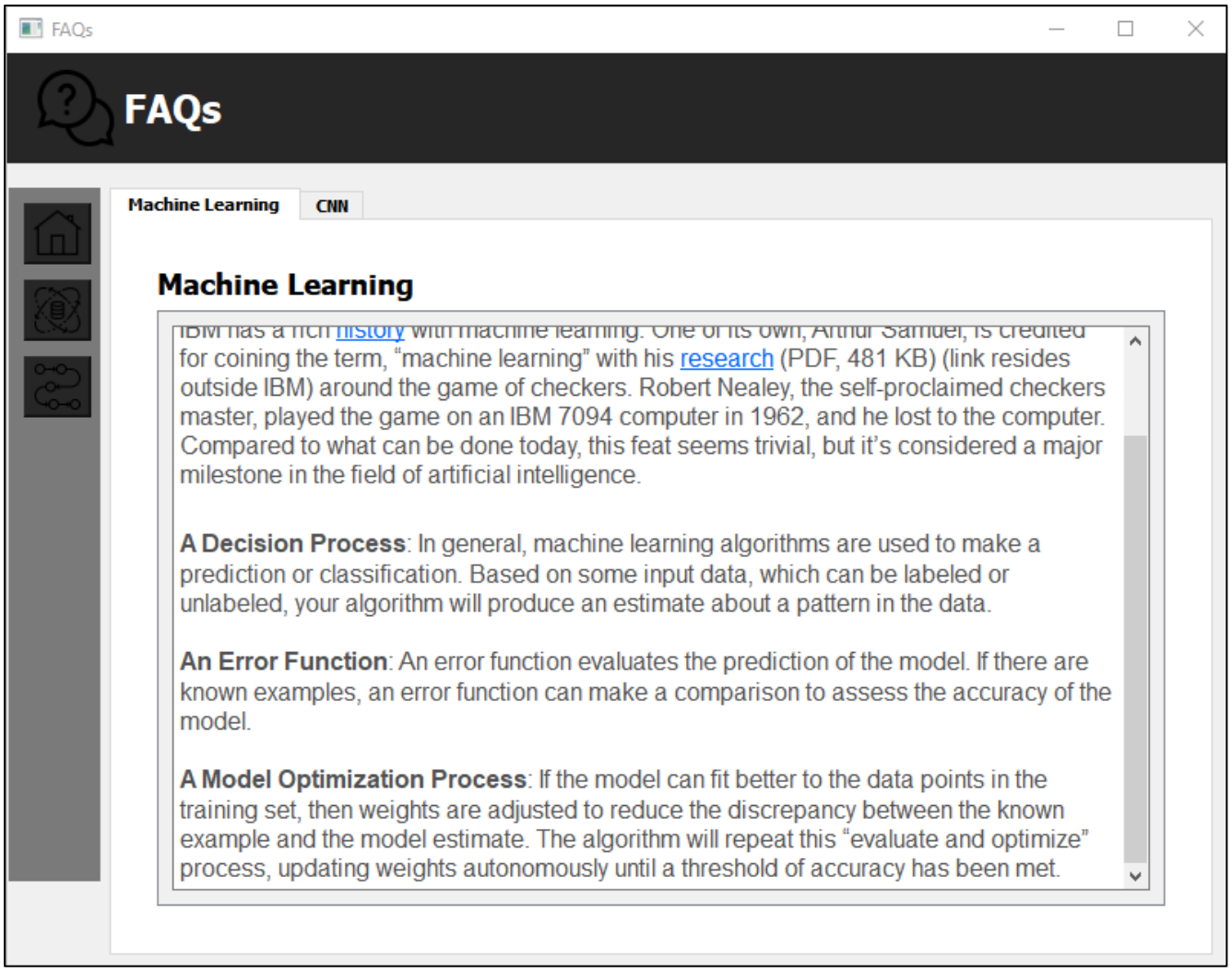


Figura 70 Vista de FAQS

Capítulo 9
Conclusiones

A lo largo del desarrollo de este TFG he tenido la oportunidad de profundizar en el conocimiento de las redes de Aprendizaje profundo y en cómo darles uso resolviendo problemas de actualidad, como es la autenticación de usuarios mediante datos de ponibles. Aunque no soy alumno de la mención de computación la elección de este TFG se ha basado en el interés especial que tengo hacia el ámbito de la *IA* y del Machine Learning.

El problema propuesto ha supuesto un reto personal para mí, ya que he encontrado muchos contratiempos, y no siempre ha sido fácil desarrollar y cumplir con los objetivos de cada Sprint. A pesar de que como alumno de la mención de diseño software no he cursado la asignatura de *Mineria de Datos*, el desarrollo de este TFG me ha permitido adquirir conocimientos prácticos sobre procesamiento de datos y desarrollo de modelos que son exportables en mi carrera profesional.

Otro aspecto importante a destacar es la elección de la metodología utilizar por parte del tutor. Aunque a lo largo de la carreara se han viso diferentes metodologías, siempre se ha utilizado un enfoque más tradicional. El beneficio principal del uso de una metodología como es SCRUM es la transparencia en el avance y objetivos del sprint, lo que favorece que haya una constancia en el proyecto. Otro beneficio de haber trabajado siguiendo una metodología ágil es que me ha permitido ver un enfoque diferente de trabajo, que me permitirá adaptarme a los estándares cambiantes del entorno laboral.

El proyecto desarrollado cuenta con 4 objetivos principales, que se han alcanzado con éxito. *El primer objetivo*, procesar y adaptar los datos ponibles, corresponde al desarrollo del Sprint 1. Este sprint en concreto ha sido uno de los que más tiempo me ha llevado, debido a que se ha necesitado de tiempo para poder comprender la naturaleza de los datos, como tratarlos y que forma tienen que tener para que puedan ser aceptados como entrada de los modelos. Para cumplir con este objetivo se han desarrollado funciones que sirven para preprocesar, guardar e importar los datos.

El segundo objetivo, desarrollar un marco de pruebas basado en redes neuronales, ofrece la funcionalidad principal de nuestro proyecto. Y corresponde al desarrollo de los Sprints 2 y 3. En primera instancia en el Sprint 2 se han creado modelos que posteriormente en el Sprint siguiente nos han servido como base para nuestro marco de pruebas. Para cumplir con los objetivos de cada Sprint se han desarrollado objetos y métodos específicos que permiten la creación e implementación del marco de pruebas.

Como tercer objetivo, se ha desarrollado una aplicación, en un entorno de desarrollo, que corresponde al Sprint 4 de nuestro proyecto. Esta plataforma, implementa la funcionalidad necesaria para hacer uso del marco de pruebas con los modelos seleccionables creados y obtener los resultados individuales de cada ejecución. Para cumplir con los objetivos se han utilizado técnicas de diseño, y patrones como el MVC o modelo vista controlador para la elaboración de la aplicación.

Como ultimo objetivo se ha propuesto que el programa se fácilmente migrable de un entorno de desarrollo a otro, para cumplir este objetivo se ha suministrado un manual de instalación para la descarga y ejecución del proyecto.

Para cumplir los objetivos de este TFG se han utilizado contenidos de varias asignaturas cursadas a lo largo de la carrera:

- Planificación y Gestión de Proyectos y Fundamentos de Ingeniería de Software: se destaca la gestión del proyecto y el diseño del sistema.
- Interacción Persona Computador, para la gestión de sistemas con interacción y aplicaciones, así como el patrón MVC y el diseño gráfico de la apariencia de las vistas.
- Fundamentos de Programación: básico para cualquier proyecto de software.
- Programación Orientada a Objetos, para la construcción de clases necesarias para la implementación de nuestro marco de pruebas.
- Fundamentos de Inteligencia Artificial, Ingeniería del Conocimiento y Técnicas de Aprendizaje Automático que han sentado las bases del proyecto y su funcionamiento.

En cuanto a la planificación inicial propuesta para la organización del proyecto, esta sufrió cambios debido a los riesgos descritos en la sección de planificación. Destacamos la dificultad en la comprensión de los problemas a resolver y la estimación incorrecta estimación del esfuerzo de algunas épicas. Otro factor que retraso el proyecto fueron las circunstancias laborales que he tenido, que no me han permitido seguir en algunas ocasiones la planificación del sprint.

9.1. Posibles mejoras

Se propone como mejora para trabajo futuro:

- Añadir más parámetros seleccionables a los modelos presentados.
- Añadir nuevos modelos de IA a la plataforma de pruebas.
- Añadir threads en la ejecución de los modelos y en el procesamiento de los datos para reducir el tiempo de ejecución de estos.
- Desarrollo de una aplicación más completa, permitiendo comparar las diferentes ejecuciones de los diferentes modelos entre sí, manteniendo un historial de estos.

Manual de instalación

Lo primero que debemos hacer es obtener el proyecto, para ello se descarga la carpeta Deliverable del repositorio de Google Drive:

https://drive.google.com/drive/folders/17orH1ZRjes0c3rDKKRHryJ4QHGdze6eh?usp=sharing

Una vez descargado, abrimos la interfaz de pycharm y seleccionamos el proyecto descargado.

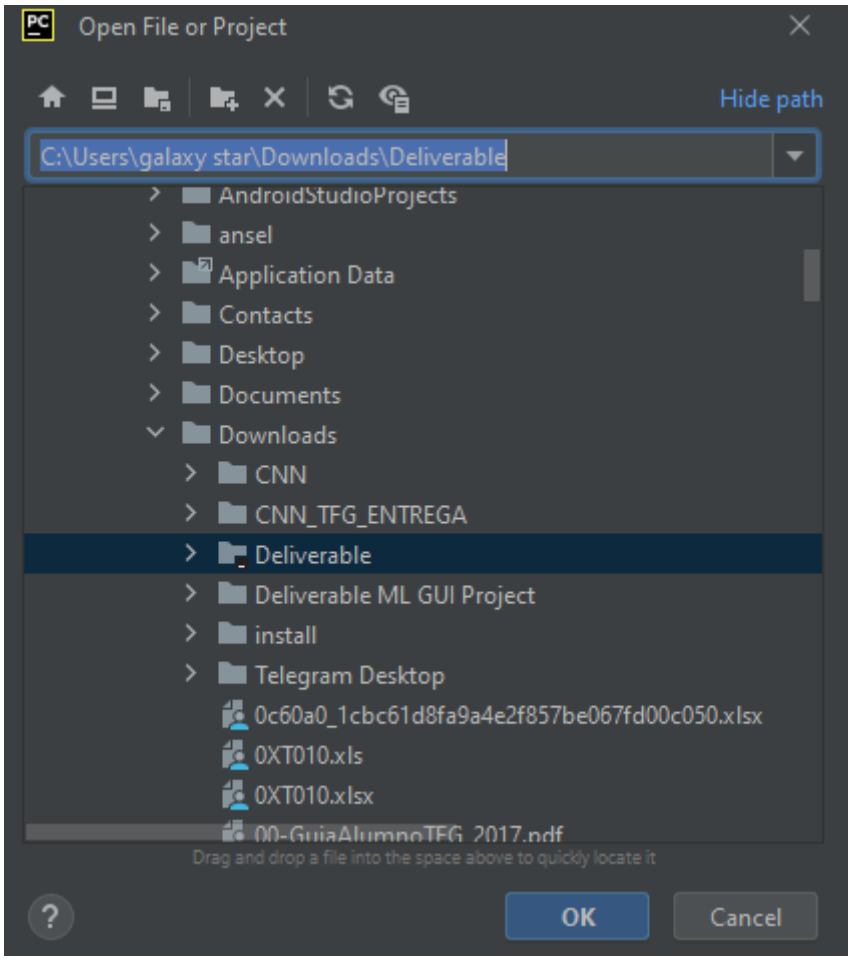


Figura 71 Ventana de abrir proyecto

Al importar el proyecto se nos pedirá que descarguemos las dependencias del proyecto y aparecerá la siguiente ventana.

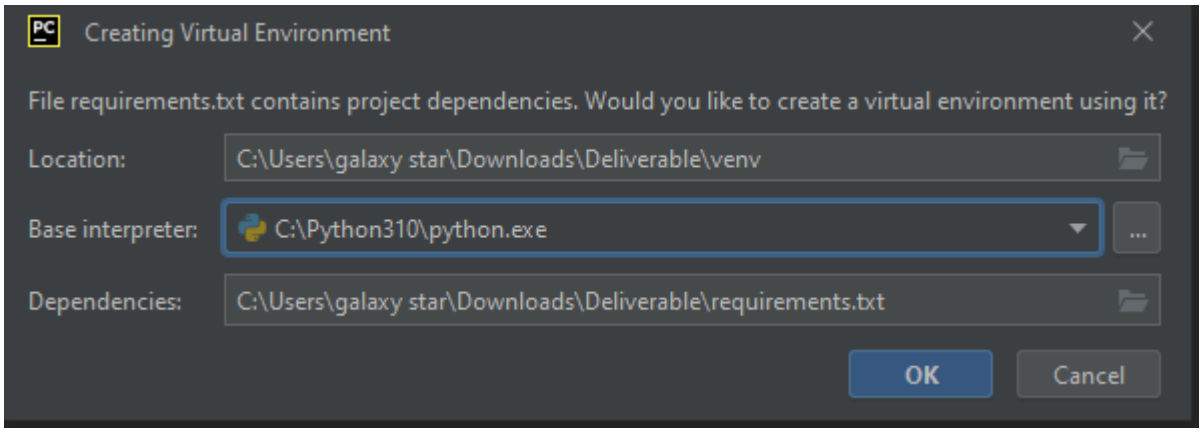


Figura 72 Ventana de creacion de entorno

Una vez se instalen las dependencias, ejecutamos main.py en el botón play de arriba a la derecha, tal como se indica en la imagen.

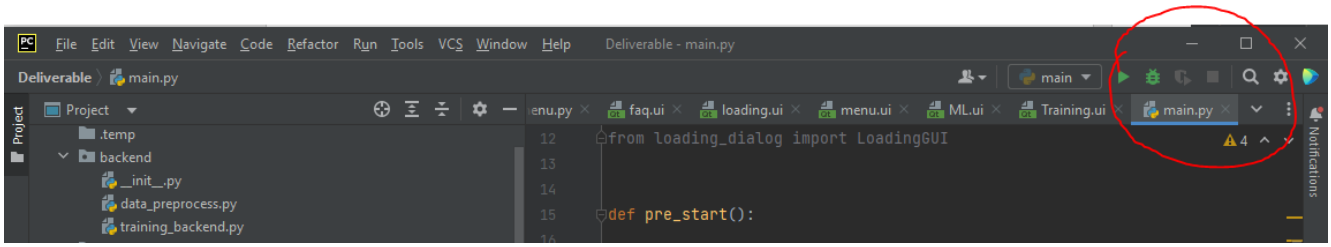


Figura 73 Instrucciones de ejecución de main.py

Bibliografía

[1] Agostinelli F, Hoffman M, Sadowski P, Baldi P. Learning activation functions to improve deep neural networks. arXiv preprint arXiv:1412.6830. 2014 Dec 21.

[2] Bershad N. Analysis of the normalized LMS algorithm with Gaussian inputs. IEEE Transactions on Acoustics, Speech, and Signal Processing. 1986 Aug;34(4):793-806.

[3] Bethge, M. y Pawelzik, K. (2003). El lenguaje de las neuronas. Mente y Cerebro. 2: 72- 79.

[4] David E. Rumelhart, Geoffrey E. Hinton y Ronald J. Williams. "Learning Representations by Back-propagating Errors". En: Nature 323.6088 (1986), págs. 533-536. doi: 10.1038/323533a0. url: http://www.nature.com/articles/323533a0.

[5] D. H. Hubel y T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex", The Journal of Physiology, vol. 148, n.o 3, p´ags. 574-591, oct. de 1959. doi: 10.1113/jphysiol.1959.sp006308. dirección: https://doi.org/10.1113/ jphysiol.1959.sp006308.

[6] E. Bayhan, Z. Ozkan, M. Namdar and A. Basgumus, "Deep Learning Based Object Detection and Recognition of Unmanned Aerial Vehicles," 2021 3rd International Congress on Human-Computer Interacti-on, Optimization and Robotic Applications (HORA), 2021, pp. 1-5, doi: 10.1109/HORA52670.2021.9461279.

[7] El Objetivo de Producto, scrum.org, drieccion: https://www.scrum.org/resources/blog/el-objetivo-de-producto. último acceso: junio 2022

[8] Google Colaboratory FAQ, Google, dirección: https://research.google.com/colaboratory/faq.html último acceso: junio 2022

[9] Guía docente trabajo de fin de grado, Universidad de Valladolid, dirección: https://albergueweb1.uva.es/guia_docente/uploads/2021/545/46976/1/Documento.pdf último acceso: junio 2022

[10] Hastie, Trevor. Tibshirani, Robert. Friedman, Jerónimo. Los elementos del aprendizaje estadístico: minería de datos, inferencia y predicción. Springer, Nueva York, NY, 2009.

[11] Haykin, Simon (1998). Neural Networks: A Comprehensive Foundation (2 ed.). Prentice Hall. ISBN 0-13-273350-1.

[12] Hussain M, Bird JJ, Faria DR. A study on cnn transfer learning for image classification. InUK Workshop on computational Intelligence 2018 Sep 5 (pp. 191-202). Springer, Cham.

[13] Irene Salvador-Ortega, Carlos Vivaracho-Pascual and Arancha Simon-Hurtado, "A Successful Study and Effective System Proposal for Biometric User Authentication by Means of Commercial Wearables" IEEE TRANSACTION ON MOBILE COMPUTING

[14] Jupyter notebook, dirección: https://docs.jupyter.org/en/latest/use/using.html último acceso: junio 2022

[15] K. Fukushima, “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”, Biological Cybernetics, vol. 36, n.o 4, p´ags. 193-202, abr. de 1980. doi: 10.1007/bf00344251. dirección: https://doi.org/10.1007/bf00344251. último acceso: junio 2022

[16] M. C. Sorkun, A. E. Danişman and Ö. D. İncel, "Human activity recognition with mobile phone sensors: Impact of sensors and window size," 2018 26th Signal Processing and Communications Applications Conference (SIU), 2018, pp. 1-4, doi: 10.1109/SIU.2018.8404569.

[17] Metodologia de gestión de proyectos, La universidad en internet Unir, https://www.unir.net/empresa/revista/metodologias-gestion-proyectos, último acceso: junio 2022

[18] Minsky, M., & Papert, S. (1969). Perceptrons. M.I.T. Press.

[19] Mon Alicia, Estayno Marcelo G, Scalzone Patricia, Definición de un marco de trabajo para la implementación inicial de un Modelo de Proceso Software (2006) dirección: http://sdici.unlp.edu.ar/handle/10915/22121

[20] Number of smartwatch users worldwide from 2017 to 2026, Statistica, dirección:https://www.statista.com/forecasts/1314339/worldwide-users-of-smartwatches, último acceso: junio 2022

[21] Ritika Wason, Deep learning: Evolution and expansion, Cognitive Systems Research,Volume 52,2018 ,Pages 701-708, ISSN 1389-0417, https://doi.org/10.1016/j.cogsys.2018.08.023. dirección https://www.sciencedirect.com/science/article/pii/S1389041717303546, último acceso: junio 2022

[22] S. Gervais-Ducouret, "Next smart sensors generation," 2011 IEEE Sensors Applications Symposium, 2011, pp. 193-196, doi: 10.1109/SAS.2011.5739775.

[23] Song, Hyoung-Kyu and AlAlkeem, “Deep User Identification Model with Multiple Biometrics” arXiv, 2019, doi 10.48550/ARXIV.1909.05417, dirección https://arxiv.org/abs/1909.05417 último acce-so: junio 2022

[24] Vincent Cheung and Kevin Cannons, “ An Introduction of Neural Networks”, Manitoba, Canada, May 27, 2002

[25] Warren McCulloch y Walter Pitts. “A Logical Calculus of Ideas Immanent in Nervous Activity”. En: Bulletin of Mathematical Biophysics 5 (1943), págs. 127-147.

[26] Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University, 1975. dirección: https://books.google.es/ books?id=z81XmgEACAAJ.

[27] Y. LeCun, P. Haffner, L. Bottou e Y. Bengio, “Object Recognition with GradientBased Learning”, en Shape, Contour and Grouping in Computer Vision, Springer Berlin Heidelberg, 1999, págs. 319-345. doi: 10.1007/3-540-46805-6_19

[28] Y. Su, T. Li, D. Wang, Y. Liu and D. Qi, "The Modeling Ability and Its Effectivity for Multi-layer ADA-LINE NN," 2009 Fifth International Conference on Natural Computation, 2009, pp. 449-453, doi: 10.1109/ICNC.2009.700.

Anexo

Se especifican aquí las versiones de los distintos paquetes que se requieren para el correcto funcionamiento de la aplicación desarrollada:

absl-py==1.1.0
astunparse==1.6.3
cachetools==5.2.0
certifi==2022.6.15
charset-normalizer==2.0.12
cycler==0.11.0
daal==2021.6.0
daal4py==2021.6.3
flatbuffers==1.12
fonttools==4.33.3
gast==0.4.0
google-api-core==2.8.2
google-auth==2.8.0
google-auth-oauthlib==0.4.6
google-cloud==0.34.0
google-cloud-vision==2.7.3
google-pasta==0.2.0
googleapis-common-protos==1.56.3
grpcio==1.47.0
grpcio-status==1.47.0
h5py==3.7.0
idna==3.3
importlib-metadata==4.11.4
joblib==1.1.0
keras==2.9.0
Keras-Preprocessing==1.1.2
kiwisolver==1.4.3
libclang==14.0.1
Markdown==3.3.7
matplotlib==3.5.2
numpy==1.22.4
oauthlib==3.2.0
opt-einsum==3.3.0
packaging==21.3
pandas==1.4.2
Pillow==9.1.1
proto-plus==1.20.6
protobuf==3.19.4
pyasn1==0.4.8
pyasn1-modules==0.2.8

pyparsing==3.0.9
PyQt5==5.15.7
PyQt5-Qt5==5.15.2
PyQt5-sip==12.11.0
PyQt5-stubs==5.15.6.0
python-dateutil==2.8.2
pytz==2022.1
requests==2.28.0
requests-oauthlib==1.3.1
rsa==4.8
scikit-learn==1.1.1
scikit-learn-intelex==2021.6.3
scipy==1.8.1
six==1.16.0
tbb==2021.6.0
tensorboard==2.9.1
tensorboard-data-server==0.6.1
tensorboard-plugin-wit==1.8.1
tensorflow==2.9.1
tensorflow-estimator==2.9.0
tensorflow-io-gcs-filesystem==0.26.0
termcolor==1.1.0
threadpoolctl==3.1.0
typing-extensions==4.2.0
urllib3==1.26.9
Werkzeug==2.1.2
wrapt==1.14.1
zipp==3.8.0