Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería del Software

Identificación de Personas mediante Redes Neuronales a partir de Datos Obtenidos por Sensores

---

Alumno: Alfredo Fernández Guaza

Tutor: Jesús M. Vegas Hernández

A mi hermano,

por ser el mejor ejemplo a seguir
y hacerme salir de mi zona de confort.

Agradecimientos

Me gustaría agradecer principalmente, a mi familia por aguantarme cada día y por ayudarme en los momentos más difíciles. Agradecer concretamente a mi hermano, Jesús Fernández Guaza, por poner el listón muy alto y ser un modelo a seguir.

A mis amigos de Palencia, por los momentos, las risas y anécdotas que me habéis brindado aunque últimamente me haya distanciado del grupo.

A la asignatura Profesión y Sociedad por ser la inspiración de la creación del grupo Pystacho y a todos los integrantes de este grupo, por los momentos vividos durante el grado y por las risas que nos hemos echado.

A Édgar Díez Alonso por ayudarme a afianzar conceptos sobre el aprendizaje automático y a Óscar Aragón Esteban por todas las tardes de *discord*.

A mi tutor, por hacer interesantes todas las asignaturas que imparte y por darme la oportunidad de realizar este TFG, en el cual he aprendido un montón sobre el aprendizaje profundo.

Gracias a todos.

Resumen

¿Crees que se puede identificar a una persona por su forma de abrir una puerta? En este Trabajo Fin de Grado, mediante técnicas de Inteligencia Artificial se explicará cómo se ha llegado a una precisión de 88 personas identificadas correctamente de cada 100.

Para ello, a partir de un conjunto de datos formado por secuencias de lecturas de sensores por cada individuo, se logra identificar a usuarios entre 47 individuos distintos por su forma de abrir la puerta mediante técnicas de Aprendizaje Profundo.

Aparte, se explica el funcionamiento de las redes neuronales utilizadas para el Aprendizaje Profundo y se dará una guía para resolver problemas de Aprendizaje Automático.

Para implementar la solución se ha empleado el lenguaje de programación *Python* junto con la biblioteca *TensorFlow* y la API *Keras*.

Palabras clave: Aprendizaje profundo, Redes Neuronales, Identificación de personas, Aprendizaje Automático, Tensorflow, Keras.

Abstract

Do you think a person can be identified by the way he/she opens a door? In this Final Degree Project, using Artificial Intelligence techniques, it will be explained how an accuracy of 88 people correctly identified out of 100 has been reached.

This was achieved using a dataset formed by sequences of sensor readings for each individual, it is possible to identify users among 47 different individuals by their way of opening the door by Deep Learning techniques.

In addition, it will be explained how neural networks used for Deep Learning works and a guide to solve Machine Learning problems will be given.

To implement the solution, the *Python* programming language has been used together with the *TensorFlow* library and the *Keras* API.

Keywords: Deep Learning, Neural Networks, Person Identification, Machine Learning, Tensorflow, Keras.

Índice general

Agradecimientos **III**

Resumen **V**

Abstract **VII**

Lista de figuras **XV**

Lista de tablas **XIX**

1. Introducción **1**

1.1. Contexto 1

1.2. Motivación 2

1.3. Objetivos 2

1.4. Estado del Arte 3

1.5. Estructura de la memoria 4

2. Aprendizaje Automático **5**

2.1. Aprendizaje Supervisado 5

2.1.1. Algoritmos de Clasificación 5

2.1.2. Algoritmos de Regresión 7

2.1.3. Algoritmos para Regresión y Clasificación 7

2.2. Aprendizaje no Supervisado 10
2.2.1. Agrupamiento o *Clustering* 10
2.2.2. Reglas de Asociación 10
2.2.3. Reducción de la Dimensionalidad 11
2.3. Aprendizaje por Refuerzo 11
2.4. Tensores 12
2.5. Conceptos Fundamentales 13
2.5.1. Evaluación de un Modelo 13
2.5.2. Preprocesado de Datos 14
2.5.3. Ingeniería de Características 14
2.5.4. Sobreajuste y Subajuste 15

3. Aprendizaje Profundo **17**

3.1. Introducción al Aprendizaje Profundo 17
3.1.1. ¿Cómo funciona el aprendizaje profundo? 17
3.1.2. ¿Cuando utilizar Aprendizaje Profundo? 18
3.2. Neurona Artificial 19
3.2.1. Perceptrón 19
3.2.2. Funciones de Activación 20
3.3. Redes Neuronales Artificiales 23
3.3.1. Redes Neuronales Convolucionales 24
3.3.2. Redes Neuronales Recurrentes 26
3.3.3. Activación *Softmax* 27
3.4. Aprendizaje de las Redes Neuronales 29
3.4.1. Función de Pérdida 29
3.4.2. Descenso del Gradiente 30
3.4.3. *Feedforward* y *Backpropagation* 32

3.5. Técnicas Avanzadas 34
3.5.1. Transferencia de Aprendizaje 34
3.5.2. Conceptos Importantes 35

4. Requisitos y Planificación **39**

4.1. Desarrollo en Cascada 39
4.1.1. Artefactos 40
4.1.2. Fases 40
4.2. Requisitos 41
4.3. Riesgos 42
4.3.1. Lista de Riesgos 42
4.3.2. Prioridades de los Riesgos 48
4.4. Planificación 50
4.4.1. Planificación de Alto Nivel 50
4.4.2. Planificación Detallada 51
4.5. Presupuesto 54
4.6. Seguimiento 55
4.6.1. Tiempo Real 55
4.6.2. Dificultades 58
4.6.3. Coste Real 58

5. Tecnologías utilizadas **59**

5.1. Gestión del Proyecto 59
5.1.1. Planificación y Comunicación 59
5.1.2. Documentación 60
5.1.3. Almacenamiento de los Cuadernos *Jupyter* 61
5.2. Redes Neuronales 62

5.2.1. Python 62
5.2.2. Tensorflow 62
5.2.3. Keras 63
5.3. Otras Tecnologías 64
5.3.1. Linux 64
5.3.2. Anaconda 64
5.3.3. Jupyter Notebook y Jupyter Lab 64
5.3.4. Google Colaboratory 65

6. Identificación de Personas **67**

6.1. Punto de Partida 67
6.1.1. Artículo 67
6.1.2. *Dataset* 68
6.1.3. Código de Partida 69
6.2. Progreso 73
6.2.1. Análisis 73
6.2.2. Diseño 74
6.2.3. Preprocesado de los Datos 74
6.2.4. Creación del Modelo 76
6.2.5. Evaluación del Modelo 81
6.2.6. Comparativas 83
6.3. Solución Final 88
6.3.1. Datos Elegidos 88
6.3.2. Modelo Final 90
6.3.3. Evaluación del Modelo 93

7. Conclusiones **95**

7.1. Problema Planteado 95

7.2. Planificación 96
7.3. Experiencia personal 96
7.4. Líneas de trabajo futuras 96

Bibliografía **97**

A. Manuales **101**

A.1. Manual de Instalación y Uso 101
A.2. Manual de Mantenimiento 103
A.2.1. Preprocesado de datos 103
A.2.2. Modelo 104
A.2.3. Evaluación 104

B. Resumen de enlaces adicionales **105**

Lista de Figuras

2.1. Árboles de decisión de clasificación y de regresión 8

2.2. Proceso del algoritmo *Support Vector Machines* [1] 9

3.1. Funcionamiento del aprendizaje profundo 18

3.2. Funcionamiento de un perceptrón 19

3.3. Función de Paso [2] 20

3.4. Función Sigmoidea [2] 21

3.5. Función ReLU [2] 21

3.6. Función tanh [2] 22

3.7. Red Neuronal Artificial [3] 23

3.8. Jerarquía de patrones [4] 24

3.9. Convolución visual [4] 25

3.10. Convolución real 25

3.11. Convolución con stride y padding [5] 26

3.12. Diagrama de una capa recurrente simple [4] 27

3.13. Descenso del Gradiente [6] 31

3.14. Diferentes formas de usar modelos pre-entrenados [7] 34

3.15. Adecuada tasa de aprendizaje [8] 36

3.16. Antes y después de aplicar dropout [9] 36

3.17. Parada temprana del entrenamiento de la red neuronal [10] 37

4.1. Matriz de riesgos - Probabilidad/Impacto 42
4.2. Planificación de alto nivel del TFG 50
4.3. Planificación detallada - Introducción del TFG 51
4.4. Planificación detallada - Planificación del TFG 52
4.5. Planificación detallada - Análisis del Sistema del TFG 52
4.6. Planificación detallada - Tecnologías Utilizadas del TFG 53
4.7. Planificación detallada - Implementación del TFG 53
4.8. Planificación detallada - Manuales del TFG 54

5.1. Microsoft Project [11] 60
5.2. Microsoft Teams [12] 60
5.3. Overleaf [13] 61
5.4. GitHub [14] 61
5.5. Google Drive [15] 61
5.6. Python [16] 62
5.7. TensorFlow [17] 62
5.8. Keras [18] 63
5.9. Linux [19] 64
5.10. Anaconda [20] 64
5.11. Jupyter Project [21] 65
5.12. Google Colaboratory [22] 65

6.1. Ejes de la manilla [23] 68
6.2. Lectura de sensores de cada muestra 71
6.3. Todas las lecturas de los sensores por cada muestra 76
6.4. Saber si el modelo sufre de *overfitting* o *underfitting* por el *loss* [24] 78

6.5. Hallar los TP,TN, FP y FN de la matriz de confusión [25] 81

6.6. Gráfica comparativa entre modelos 85

6.7. Acelerómetro X 86

6.8. Giroscopio Z 87

6.9. Acelerómetro X y giroscopio Z 87

6.10. Topología de la red neuronal del modelo A 90

6.11. Matriz de Confusión - Modelo Final 93

6.12. Diagrama de caja y bigotes - Modelo final 94

A.1. Subir el *dataset* 102

A.2. Ejecutar todo el cuaderno 102

A.3. Abrir el *dataset* 103

Lista de Tablas

4.1. Resumen Planificación 55
4.2. Seguimiento - Introducción 55
4.3. Seguimiento - Planificación 56
4.4. Seguimiento - Análisis del Sistema 56
4.5. Seguimiento - Tecnologías Utilizadas 57
4.6. Seguimiento - Implementación 57
4.7. Seguimiento - Manuales 57

6.1. Tabla comparativa entre modelos 85

Capítulo 1

Introducción

En este capitulo se introducirá el tema escogido en este TFG, la razón por la cual se ha elegido, los principales objetivos de este proyecto, proyectos similares a este y la estructura que seguirá la documentación.

1.1. Contexto

Este Trabajo Fin de Grado trata de la continuación de un proyecto realizado por mi tutor Jesús M. Vegas Hernández. Esté proyecto utiliza técnicas de Inteligencia Artificial, concretamente un conjunto de algoritmos de Aprendizaje Automático llamado Aprendizaje Profundo o *Deep Learning* en inglés.

Se hará uso de redes neuronales convolucionales y recurrentes para resolver un problema de clasificación. El problema consiste en clasificar a individuos por la forma en que abren una puerta a través de una manilla. Para ello Jesús M. Vegas y César Llamas colocaron sensores y extrajeron sus datos en un *dataset*, un conjunto de datos, que posteriormente fueron tratados para introducirlos en la red neuronal.

Se explicará el proceso que se ha seguido a la hora de solucionar el problema propuesto, así como las métricas con las que se ha evaluado las distintas propuesta y haciendo una comparativa entre estas. Para lograr entender como se ha solucionado el problema se presentará un marco teórico sobre aprendizaje profundo y sobre conceptos fundamentales del aprendizaje automático.

Además de la parte técnica, en este proyecto se llevará a cabo una planificación y un seguimiento de está. Al final llegaré a una conclusión sobre el proyecto y comentaré donde se puede encontrar el código, como ejecutarlo y como se deben hacer las modificaciones del propio código.

1.2. Motivación

El principal motivo de este trabajo es tener una herramienta que permita hacer un seguimiento de las personas que entran y salen en una habitación ayudado por otros métodos de identificación. Puede ser útil a las empresas en el caso de que sus empleados fichen físicamente.

También puede tener finalidades educativas, por ejemplo se puede controlar la asistencia de un grupo de alumnos con esta herramienta, un poco de hardware y software adicional.

Otro motivo secundario es el de adquirir conocimientos sobre el Aprendizaje Profundo, siendo este trabajo un primer contacto con el *Deep Learning*.

1.3. Objetivos

El objetivo principal de este Trabajo Fin de Grado es analizar y probar que configuración de redes neuronales es la óptima para identificar al personal a partir de los datos obtenidos por sensores que trató mi tutor Jesús M. y César Llamas. Para ver que configuración es la óptima debemos ver cual de todas es la que maximiza la precisión o lo que es lo mismo, el porcentaje de cuantas personas se han identificado correctamente de las personas que se han identificado.

Para conseguir este objetivo será necesario cumplir las siguientes metas:

1. Leer el libro Deep Learning with Python de François Chollet[4] para aprender sobre el aprendizaje profundo y así poder abordar el problema principal. Tras esta lectura obtendré los conocimientos necesarios sobre los distintos tipos de redes neuronales y sabré que tipo de red neuronal o que combinación tipos de redes neuronales es la apropiada.
2. Realizar un estudio de que configuración/es de la red neuronal es la apropiada para el tipo/s de red/es neuronal/es. Medible mediante el número de páginas de estudio realizadas.
3. Implementar y después analizar la/s configuración/es de la red neuronal, ajustándola/s si fuera necesario, para obtener finalmente una configuración óptima.

Como objetivo secundario de este proyecto tendré que realizar un estudio teórico sobré el aprendizaje automático y el aprendizaje profundo por si no podemos cumplir el objetivo principal.

1.4. Estado del Arte

Investigando trabajos de fin de grado de otros años, he encontrado que en muchos abordan el tema de las redes neuronales, aplicando estos algoritmos de inteligencia artificial a problemas específicos.

No he encontrado ningún TFG sobre aprendizaje profundo que también introduzca el aprendizaje automático, ni aborde la posible resolución específica de el problema que se va a estudiar en este proyecto.

La mayoría de los trabajos sobre *deep learning* investigan sobre la resolución de un problema relacionado con la clasificación de imágenes o vídeos o regresión de resultados a partir de imágenes o vídeos, no he encontrado ninguno que trate sobre la investigación de un problema cuyos datos de entrada sean secuencias de datos obtenidos de sensores.

Entre los ejemplos de TFGs de alumnos anteriores sobre aprendizaje profundo, destacar el trabajo de Jorge Barrio Conde.

- Análisis y Experimentación Práctica de Frameworks Deep Learning Aplicados a la Astronomía [26].
- Deep Learning para Análisis de Forma en Fabricación Automática [27].
- Caracterización de piezas mediante técnicas de Deep Learning [28].
- Análisis de entrenamiento en deep learning para gestión de calidad en fabricación automática [29].

1.5. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 2 - Aprendizaje Automático: Describe el marco teórico sobre el aprendizaje automático y como abordar un problema mediante algoritmos de *machine learning*.

Capítulo 3 - Aprendizaje Profundo: Describe el marco teórico sobre el aprendizaje profundo, desde conceptos fundamentales como una simple neurona artificial hasta tipos de redes neuronales.

Capítulo 4 - Requisitos y planificación: Describe el ciclo de vida utilizado para el proyecto y como se ha adaptado este proyecto a dicho. Además de la lista de requisitos, el análisis de riesgos, la planificación y el presupuesto del proyecto. Finalmente, se comentará el seguimiento del proyecto.

Capítulo 5 - Tecnologías utilizadas: Describe las tecnologías utilizadas tanto para la gestión de la documentación del Trabajo Fin de Grado como para el desarrollo del proyecto.

Capítulo 6 - Identificación de Personal: Describe el trayecto de creación del producto final que va a ser entregado.

Capítulo 7 - Conclusiones: Describe un resumen del proyecto de forma retrospectiva sobre que se ha hecho correctamente y que no, sirviendo como reflexión del proyecto. Incluye las líneas de trabajo futuras.

Anexo A Manuales: Incluye un manual de instalación y uso, además de un manual de mantenimiento.

Anexo B Resumen de enlaces adicionales: Incluye enlaces de interés sobre el proyecto, como el repositorio del código final como del código usado para su desarrollo.

Capítulo 2

Aprendizaje Automático

El aprendizaje automático se define como el estudio de la computación de algoritmos que pueden mejorar automáticamente mediante la experiencia y el uso de datos [30].

Entre estos algoritmos se encuentran categorizados en tres grupos: el aprendizaje supervisado, el aprendizaje no supervisado, el aprendizaje por refuerzo.

2.1. Aprendizaje Supervisado

Como su propio nombre indica, se enseña a la computadora siendo supervisado por el humano. El algoritmo aprende con los pares: datos de entrada y etiquetas o respuestas correctas que permiten a los algoritmos obtener conclusiones a partir de nuevos datos y así obtener una aproximación a la etiqueta del nuevo dato.

Los principales algoritmos usados en el aprendizaje supervisado son algoritmos de clasificación y algoritmos de Regresión. Las principales diferencias residen en que los modelos de regresión se utilizan para predecir variables continuas como lo son las modas del mercado. Mientras que los algoritmos de clasificación ayudan a dividir los datos en diferentes clases. A continuación explicaremos con mayor detalle estos dos tipos de algoritmos.

2.1.1. Algoritmos de Clasificación

Consisten en el proceso de encontrar una función que ayude a dividir los datos o *dataset* en diferentes clases basándose en distintos parámetros [31]. En este tipo de algoritmos de clasificación se utiliza un programa para entrenarse utilizando los pares (datos, etiqueta) para clasificarlos en diferentes clases.

Entre los algoritmos de clasificación se encuentran: *naive bayes*, regresión logística, redes neuronales, árboles de decisiones, *random forest*, k-vecinos más cercano, *support vector machines*. A continuación explicaré brevemente cada algoritmo.

Bayesiano ingenuo o *Naive Bayes*

Fundamentado en el teorema de Bayes [32].

$$P(C|F_1, ..., F_n) = \frac{P(C) \cdot P(F_1, ..., F_n|C)}{P(F_1, ..., F_n)} \tag{2.1}$$

- $P(C|F_1, ..., F_n)$ - Probabilidad de pertenecer a una clase si presentan o ausentan las características medidas por las variables $F_1, ..., F_n$.
- $P(C)$ - Probabilidad de pertenecer a una clase.
- $P(F_1, ..., F_n|C)$ - Probabilidad de tener ciertas características si se pertenece a una clase.

Se le otorgan modelos probabilísticos que devuelvan las probabilidades de presencia o ausencia de características. Todas aquellas características $F_1, ..., F_n$ deben seguir un modelo probabilístico y deben ser independientes unas de otras.

Para clasificar la muestra, se calcula la probabilidad usando el teorema de bayes 2.1. Donde C es la variable condicionada por todas las demás características. La variable dependiente C también sigue otro modelo probabilístico que puede ser $1/clases$, siempre que las clases sean equiprobables.

Finalmente, se clasifica la muestra en el grupo con la mayor probabilidad obtenida de aplicar bayes tantas veces como clases haya. En wikipedia [32] se puede encontrar un ejemplo.

Regresión Logística o *Logistic Regression*

La regresión logística sirve para la clasificación binaria, prediciendo si la muestra de entrada pertenece o no a una clase o grupo. Para clasificar binariamente en la regresión logística se utiliza una ligera modificación de la función sigmoidea. La función sigmoidea se explicará cuando se hable sobre las funciones de activación en el capitulo de aprendizaje automático [33].

2.1.2. Algoritmos de Regresión

Consiste en el proceso de encontrar correlaciones entre variables dependientes e independientes, ayudando en la predicción de variables continuas como puede ser predecir tendencias del mercado o la predicción del tiempo. El objetivo de los algoritmos de regresión es encontrar la función que mapea la variable de entrada x a la variable continua de salida y. [31]

Entre los algoritmos de regresión se encuentran: regresión lineal simple, regresión lineal múltiple, regresión polinómica, redes neuronales, regresión del árbol de decisión, *random forest regression*, *K-nearest neighbors*, *support vector regression*.

Regresión lineal simple

Es el algoritmo de regresión mas simple de todos, para ello se haya una recta que nos indique la tenencia de un conjunto de datos continuos.

$$f(x) = a \cdot x + b \tag{2.2}$$

Regresión lineal múltiple

Parecido a la regresión lineal simple, pero en vez de tener una única variable independiente se tienen múltiples variables independientes cada una con su coeficiente.

$$f(x_1, ..., x_n) = a_1 \cdot x_1 + a_2 \cdot x_2 + ... + a_n \cdot x_n + b \tag{2.3}$$

Regresión lineal polinómica

Otro modelo de regresión, pero está vez se utiliza un polinomio para indicar la tendencia de los datos. Siendo m el número de variables independientes y n el grado del polinomio

$$f(x_1, ..., x_n) = a_{11} \cdot x_1 + a_{21} \cdot x_1^2 + ... + a_{n1} \cdot x_1^n + ... + a_{1m} \cdot x_m + a_{2m} \cdot x_m^2 + ... + a_{nm} \cdot x_m^n + b \tag{2.4}$$

2.1.3. Algoritmos para Regresión y Clasificación

Existen algoritmos que pueden utilizarse tanto para clasificar como para predecir valores, en este apartado explicaré estos algoritmos.

Redes Neuronales o *Neural Network*

Se desarrollará en profundidad en el capitulo de aprendizaje profundo. Para resumir las redes neuronales artificiales son redes de unidades de procesamiento simple que se comunican entre ellas enviando señales unas a otras a lo largo de conexiones con pesos en cada una.

También se utilizan en el aprendizaje no supervisado, además de en el aprendizaje supervisado.

Árbol de Decisión o *Decision Tree*

Árbol donde cada nodo es el momento en el que se va a tomar una decisión, las conexiones unidireccionales son representan cada acción distinta y la solución final es un vector de los nodos del camino transitado que se obtiene en función de las diversas probabilidades que tienen cada acción [34].

Dependiendo de si las variables de respuesta o los nodos hoja del árbol de decisión son continuos o numéricos en vez de categóricos.

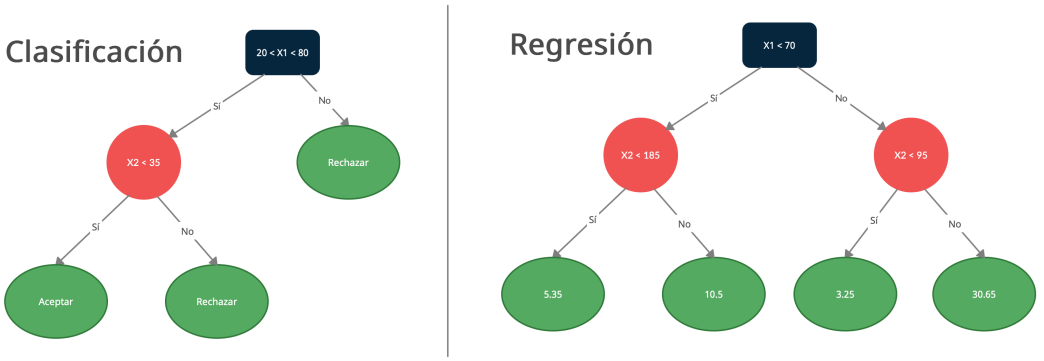


Figura 2.1: Árboles de decisión de clasificación y de regresión

Bosques Aleatorios o *Random Forest*

Combina la salida de múltiples árboles de decisión para obtener un único resultado. Cuando hay que resolver un problema de regresión con este tipo de algoritmo, se hace la medía de la salida de todos los árboles de decisión. En el caso de usarse para una tarea de clasificación se categoriza la entrada en la categoría que aparezca con más frecuencia como resultado de cada árbol de decisión [35].

K Vecinos más cercanos o *K-Nearest Neighbors*

Como su nombre indica, este algoritmo separa las muestras en grupos teniendo en cuenta su distancia a las k muestras más cercanas. El algoritmo tiene dos fases: el entrenamiento y la clasificación.

En la fase de entrenamiento, para cada muestra se almacenan los pares (vector, clase) que representan a las muestras etiquetadas [36].

En la fase de clasificación, para cada muestra sin clasificar (sin etiqueta o clase) se calcula la distancia a todos los vectores almacenados, seleccionando la distancia a las k muestras más cercanas y se clasifica la nueva muestra en la clase que más se repite en los vectores seleccionados [36].

Si se trata de resolver un problema de regresión, en vez de devolver la clase que más se repite, se retorna la media de los puntos de las k muestras más próximas al vector de la muestra de entrada.

Máquinas de vectores de soporte o *Support Vector Machines*

Funciona creando una separación entre los datos de forma que se pueda categorizar y después transformando los datos de forma que el separador sea un hiperplano facilitando la predicción de la clase de la nueva muestra. *Support Vector Machines* también puede utilizarse como algoritmo de regresión.

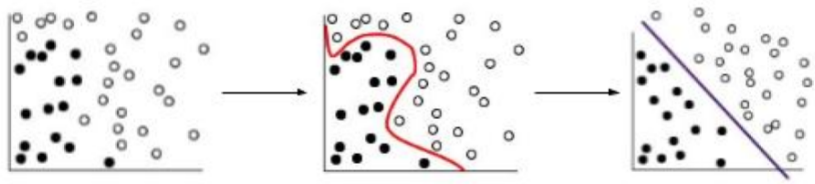


Figura 2.2: Proceso del algoritmo *Support Vector Machines* [1]

2.2. Aprendizaje no Supervisado

En este tipo de aprendizaje, no se le enseña a la computadora los cuales son las respuestas correctas. Esta vez el modelo encuentra patrones y separa los datos de entrada en distintos grupos de acuerdo con los parecidos que encuentra. Es similar a como los humanos aprendemos pensando en previas experiencias. No trabaja con datos etiquetados por lo que puede ser útil si no se dispone de datos etiquetados.

Los algoritmos de aprendizaje no supervisado se pueden categorizar en algoritmos de agrupamiento, algoritmos de asociación, algoritmos de reducción de la dimensionalidad.

2.2.1. Agrupamiento o *Clustering*

La técnica de agrupamiento o *clustering* se basa en separar en grupos datos que no están etiquetados por las similitudes o diferencias entre los propios datos [37].

Los algoritmos de de agrupamiento pueden separarse en distintas categorías:

- Exclusivos específicamente - Agrupamiento de las K-Medias.
- Superpuesto - Agrupamiento de las K-Medias.
- Jerárquico - Vinculación de Ward, Vinculación promedia, Vinculación completa y Vinculación simple.
- Probabilístico - Modelos de Mezcla Gaussiana.

2.2.2. Reglas de Asociación

Es un método basado en reglas para buscar relaciones entre las variables en un conjunto de datos o *dataset*. Se suele utilizar por ejemplo para el análisis de la cesta de la compra, permitiendo comprender mejor: las relaciones entre los diferentes productos y los hábitos de consumo [37].

Existen diferentes algoritmos utilizados para generar reglas de asociación, por ejemplo:

- *Apriori*.
- *Eclat*.
- *FP-Growth*.

El algoritmo *Apriori* es popular por ser utilizados en sistemas de recomendación. En los sistemas de recomendación se busca predecir un elemento de distintos tipos de temas de información, como pueden ser: películas, música, libros, noticias, etc.

Apriori halla una ponderación de las características de referencia del tema o temas a buscar una recomendación. Un ejemplo de algoritmo más usado para un sistema de recomendación es el llamado *Nearest Neighborhood*. Se selecciona las preferencias de los primeros N-vecinos y utilizando la correlación de *Pearson* se halla el elemento recomendado [38].

2.2.3. Reducción de la Dimensionalidad

Es una técnica utilizada cuando el número de características (*features*) o el número de dimensiones es muy alto. Para resolver este problema se reduce la cantidad de datos de entrada a un tamaño manejable, preservando la integridad del conjunto de datos o *dataset* tanto como sea posible [37]. Aparte se suele utilizar en el preprocesado de datos.

Algunos métodos de reducción de la dimensionalidad son:

- Análisis de los componentes principales.
- Descomposición del valor singular.
- Codificadores automáticos o *autoencoders*.

2.3. Aprendizaje por Refuerzo

Es un modelo parecido al aprendizaje supervisado, pero no es entrenada usando las muestras de datos. El algoritmo aprende por prueba y error. El modelo de aprendizaje por refuerzo no solo tiene capacidad de aprender a asignar a una entrada una salida, sino que además puede asignar una serie de entradas a un conjunto de salidas con dependencias [39].

Durante el aprendizaje, el algoritmo explora explora los pares (estado, acción) dentro un entorno (estado y acciones posibles en un estado específico), eligiendo la mejor acción para un determinado estado. Ese determinado estado debe conducir al estado meta [39].

2.4. Tensores

Se suelen utilizar en los modelos de redes neuronales, que se explicarán posteriormente. Los tensores matrices multidimensionales de un tipo uniforme e inmutables [40] y tienen:

- **Ejes o Dimensiones**
- **Rango:** El número de ejes o dimensiones del tensor. Dependiendo el rango del tensor estos se pueden llamar tensores 1D, tensores 2D, ... , tensores nD. Si el tensor es un escalar este se denomina tensor 0D por ser de rango 0.
- **Forma o *shape*:** El número de elementos por cada dimensión del tensor.
- **Tamaño:** el número total de elementos en el tensor.

En el mundo real, se suelen utilizar los tensores para representar los datos de las siguientes categorías [4]:

- **Datos vectorizados** - tensores 2D con la forma (muestras, características)
- **Secuencias de datos** - tensores 3D con la forma (muestras, marcas de tiempo, características)
- **Imágenes** - tensores 4D con la forma (muestras, altura, anchura, canales)
- **Vídeo** - tensores 5D con la forma (muestra, imágenes que forman el vídeo, altura, anchura, canales)

2.5. Conceptos Fundamentales

En esta sección explicaré el flujo que hay que seguir para resolver cualquier problema de aprendizaje automático y desarrollaré brevemente los pasos a seguir [4].

1. Definir los datos de entrada y la salida esperada.
2. Definir el problema: Si es de regresión, clasificación, agrupación, etc.
3. Elegir medida de éxito: Precisión si es un problema de clasificación donde cada clase es equiprobable, precisión y exhaustividad si es un problema donde las clases están desbalanceadas, etc.
4. Decidir como evaluar el modelo de aprendizaje automático: *hold-out, K-fold,* etc.
5. Preprocesar los datos.
6. Mediante ingeniería de características hacer transformaciones a los datos preprocesados necesarias para alimentar el modelo.
7. Hacer un modelo que sobreajuste o sufra de *overfitting*.
8. Modificar el modelo hasta que alcanzar un balance entre *overfitting* y *underfitting* obteniendo un buen resultado en la métrica seleccionada.

2.5.1. Evaluación de un Modelo

Un modelo de aprendizaje automático tiene como objetivo obtener buenos resultados con datos no vistos nunca antes por el modelo. Para saber si nuestro modelo generaliza correctamente es necesario conocer medidas de como de preciso es el modelo con nuevos datos.

Para poder entrenar y evaluar un modelo, es necesario que existan tres subconjuntos del *dataset* independientes entre si.

Los Subconjuntos de Entrenamiento, Validación y Evaluación

- **Entrenamiento:** Como su nombre indica son los datos utilizados en el entrenamiento del modelo.
- **Validación:** Estos datos se utilizan para comprobar que el modelo que ha sido entrenado con el subconjunto de datos de entrenamiento generaliza correctamente. Resulta que estos datos están sesgados a las modificaciones que se hagan en el modelo, debido a los sucesivos entrenamientos y validaciones con los mismos datos.
- **Evaluación:** Para eliminar este sesgo se utiliza un tercer subconjunto independiente de los dos anteriores. De este modo podemos conocer si el modelo generaliza para nuevos datos no sesgados a las modificaciones humanas en el modelo.

Diferentes Técnicas de Validación

- ***Hold-out:*** Útil cuando hay muchos datos, divide los datos en los subconjuntos de entrenamiento, validación y evaluación [4].
- ***K-fold:*** Útil cuando hay pocos datos, divide primero en dos subconjuntos uno que contiene los datos de evaluación y el otro se vuelve a dividir en K particiones del mismo tamaño. Se entrena el modelo con $K-1$ particiones y se valida con la partición restante, repitiendo el entrenamiento tantas veces como particiones haya y para cada repetición la partición con la que se valida tiene que ser distinta [4].
- ***K-fold* iterativo con mezcla:** Útil cuando hay pocos datos y necesitas una evaluación precisa de los datos. Similar a *K-fold* pero se mezclan los datos antes de dividirlo en las K particiones y se ejecuta el algoritmo de K-fold tantas veces como uno quiera [4].

2.5.2. Preprocesado de Datos

Antes de entrenar nuestro modelo debemos adecuar las entradas del mismo, ya que los *datasets* pueden tener datos inconsistentes o redundantes. Incluso pueden llegar a faltar datos. Para ello las técnicas de preprocesado ayudan a limpiar de problemas el conjunto de datos inicial.

Algunas de las técnicas más utilizadas son [41]:

- **Manejar los valores nulos:** una simple opción es eliminar las columnas o las filas que contengan datos nulos.
- **Normalización:** para tener todos los datos en una misma escala (0 - 1).
- **Manejar variables categóricas:** las categorías pueden no venir en el tipo adecuado que acepta el modelo.
- **Codificación *One-Hot*:** crea un vector del mismo tamaño que el numero de clases inicializado a 0 y asigna 1 a la posición de la clase a la que representa una muestra.

2.5.3. Ingeniería de Características

También llamado *feature engineering*, es el proceso de extraer características a partir de los datos preprocesados. Para extraer las características se usa el conocimiento que uno tiene sobre los datos y sobre el algoritmo de aprendizaje automático que se va a usar para resolver el problema [4].

La finalidad de la ingeniería de características es mejorar la calidad de los resultados al ejecutar el algoritmo de aprendizaje automático comparado a usar los datos preprocesados sin hacer ingeniería de datos [4].

Se diferencia del preprocesado de datos en que la ingeniería de características se utiliza en la codificación a mano de las transformaciones de los datos preprocesados para obtener una representación de los datos más significativa.

2.5.4. Sobreajuste y Subajuste

En este subapartado comentaré los dos problemas que más ocurren en cualquier problema de aprendizaje automático, que son el *overfitting* y *underfitting*, o en castellano, sobreajuste o subajuste.

Para entender estos conceptos es fundamental entender la diferencia entre optimización y generalización. La optimización es el proceso que ajusta el modelo para obtener el mejor rendimiento posible mientras que la generalización es el como de bien se comporta el algoritmo [4].

Existe una forma de controlar ligeramente el sobreajuste y el subajuste mediante la regularización. La regularización reduce aleatoriamente el número de características.

Subajuste o *Underfitting*

Ocurre cuando nuestro modelo es muy simple o cuando tenemos pocos datos. Al no poder optimizar el modelo, por la simplicidad de este o por la falta de datos, el modelo no generalizará como esperamos. Una forma de evitar el *underfitting* es mediante:

- Usar modelos más complejos.
- Recolectar más datos.
- Entrenar más el modelo.
- Decrementar la regularización

Sobreajuste o *Overfitting*

Ocurre cuando el algoritmo es muy complejo o cuando se le ha entrenado más de la cuenta. Al optimizar tanto el modelo, cuando probamos el algoritmo con nuevos datos, el modelo generaliza peor de lo esperado. Una forma de evitar el *overfitting* es mediante:

- Parar de entrenar justo antes de que se se produzca el sobreajuste o *overfitting*.
- Entrenar con más datos.
- Hacer más simple el modelo.
- Incrementar la regularización.

Capítulo 3

Aprendizaje Profundo

El aprendizaje profundo o *deep learning* es un conjunto de algoritmos que forman parte del aprendizaje automático o *machine learning* [42]. Estos algoritmos pueden ser de aprendizaje supervisados como no supervisados.

3.1. Introducción al Aprendizaje Profundo

El aprendizaje profundo utiliza capas sucesivas de representaciones cada vez más significativas, estas capas aprenden utilizando redes neuronales o *neural networks*. Aunque el aprendizaje profundo esté inspirado de nuestra compresión sobre el cerebro, no son modelos del cerebro.

3.1.1. ¿Cómo funciona el aprendizaje profundo?

Las capas utilizadas en el aprendizaje profundo producen transformaciones mediante sus parámetros, pesos o *weights*. El aprendizaje ocurre cuando se ajustan los valores de los pesos o parámetros de todas las capas de la red, de tal forma que relacione correctamente las muestras de entrada a sus objetivos. Dependiendo de si la tarea es de clasificación, regresión los objetivos serán clases o valores continuos. El problema reside en el número de parámetros de una red neuronal que puede ser alto [4].

Para entrenar la red neuronal formada por múltiples capas, es necesario conocer la salida obtenida tras las sucesivas transformaciones por todas las capas de la red neuronal y la salida esperada. Se calcula la función de pérdida o *loss function* de la red, mediante el computo de una puntuación que indica como de bien predice la red neuronal. La función de pérdida o también llamada función objetivo o *objective function*, computa las predicciones de la red que son correctas para obtener la puntuación [4].

La puntuación de pérdida o *loss score* se utiliza como señal de retroalimentación que sirve para ajustar los valores de los pesos o parámetros de cada capa de la red neuronal. Se ajustarán los pesos de forma que reduzca la puntuación de pérdida. El encargado de ajustar los pesos es el llamado optimizador, que está implementado mediante el algoritmo de propagación hacia atrás o *backpropagation* [4].

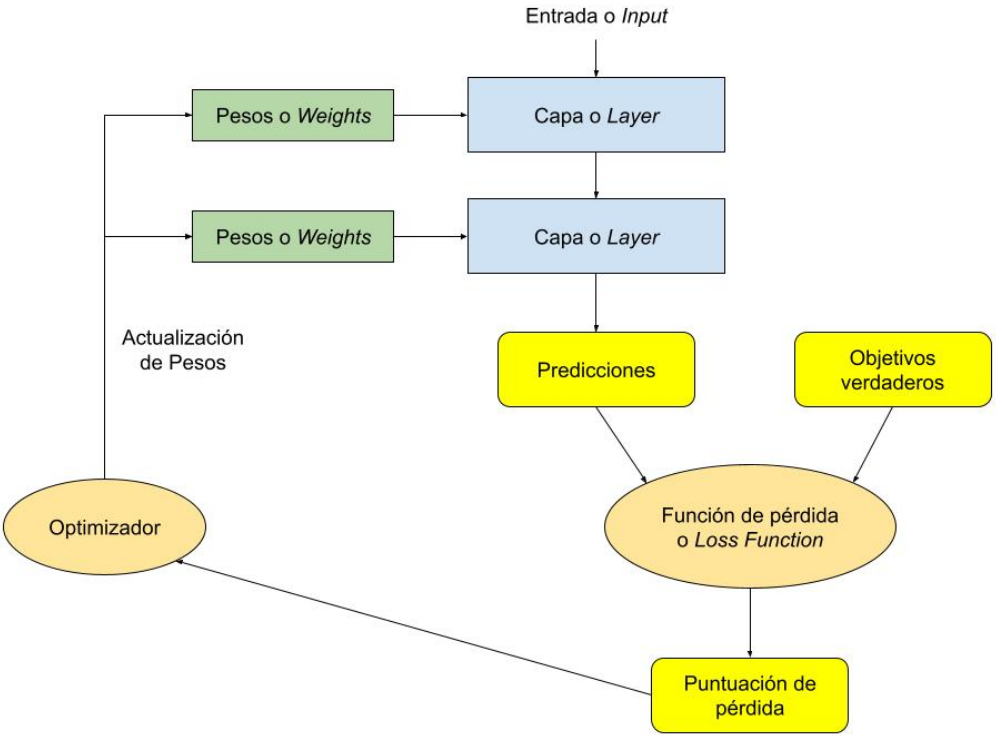


Figura 3.1: Funcionamiento del aprendizaje profundo

3.1.2. ¿Cuando utilizar Aprendizaje Profundo?

El aprendizaje profundo es ideal aplicarlo para problemas donde exista un gran número de datos. Otro caso en el que se puede aplicar *deep learning* es cuando el problema es muy complejo y es muy difícil de resolver mediante otras técnicas de aprendizaje automático [43].

Ejemplos de cuando hay que aplicar técnicas de aprendizaje profundo son la clasificación de imágenes y vídeo, el reconocimiento de voz y la conducción autónoma entre muchos otros ejemplos.

3.2. Neurona Artificial

Para aprender como funciona un modelo compuesto por múltiples redes neuronales es necesario saber de que se compone cada red neuronal, para ello en esta sección explicaré la unidad más básica de una red neuronal.

3.2.1. Perceptrón

Es la unidad básica de de una red neuronal. Intenta emular el comportamiento de una neurona biológica de forma matemática. En la imagen 3.2 se muestra el funcionamiento de un perceptrón.

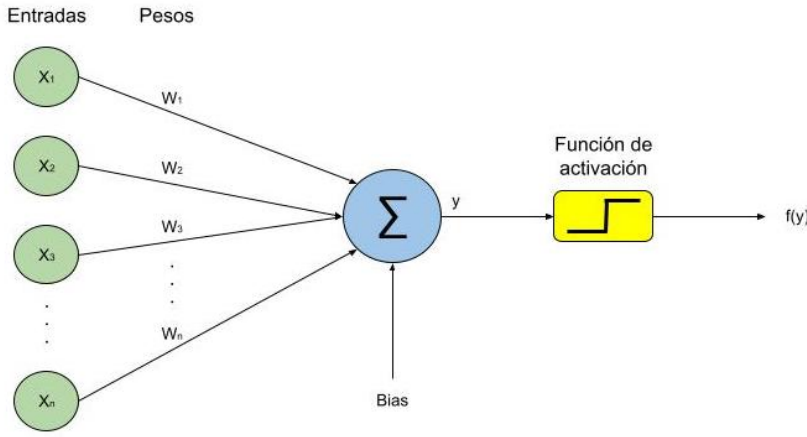


Figura 3.2: Funcionamiento de un perceptrón

Una neurona artificial que tiene n entradas, tendrá un peso por entrada. Este peso indica la relevancia de esa entrada. La salida del perceptrón se calcula aplicando una función de activación a y 3.1.

$$y = bias + \sum_{i=1}^{n} x_i \cdot w_i \tag{3.1}$$

El *bias* o sesgo, es la facilidad si el *bias* es positivo o dificultad si el *bias* es negativo con la que el valor obtenido de la función de activación devolverá un valor alto.

3.2.2. Funciones de Activación

Se utilizan para controlar la salida de un perceptrón. Dependiendo del problema, conviene usar una función de activación determinada. Entre las distintas funciones de activación están: la función de paso, la función de decisión, la función sigmoidea, la función correctora o ReLU, la función *softplus* y la función tangente hiperbólica.

Función de Paso

Devuelve 1 si la entrada es mayor que 0 y la salida será 0 en otro caso.

$$\phi_1(y) = \begin{cases} 1 & si \text{ y} > 0 \\ 0 & sino \end{cases} \tag{3.2}$$

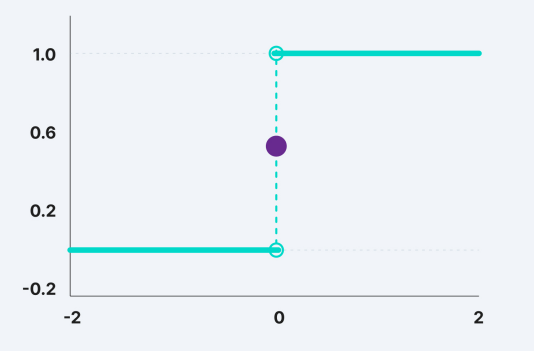


Figura 3.3: Función de Paso [2]

Función de Decisión

Devuelve 1 si la entrada es mayor que 0 y la salida será -1 en otro caso.

$$\phi_2(y) = \begin{cases} 1 & si \text{ y} > 0 \\ -1 & sino \end{cases} \tag{3.3}$$

Función Sigmoidea

Conduce a una probabilidad de valor entre 0 y 1. Se suele utilizar para clasificación o agrupación binaria.

$$\sigma(y) = \frac{1}{1 + e^{-y}} \tag{3.4}$$

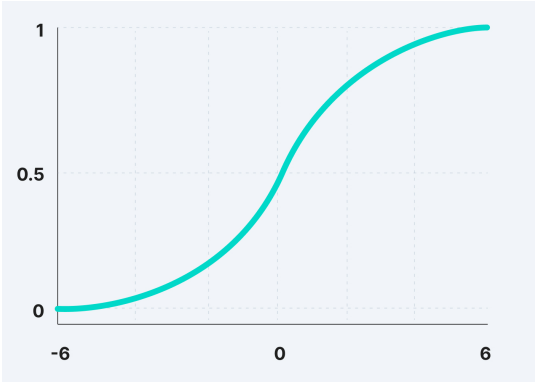


Figura 3.4: Función Sigmoidea [2]

Función Correctora o ReLU

Devuelve 0 si la entrada es menor que 0 y devuelve la entrada como salida en otro caso. Útil en las redes neuronales convolucionales.

$$ReLU(y) = max(0,\ y) \tag{3.5}$$

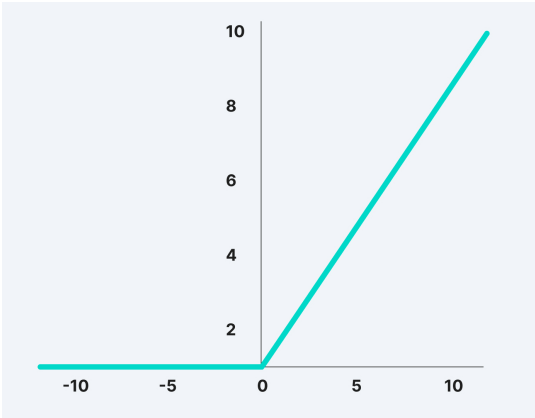


Figura 3.5: Función ReLU [2]

Función *Softplus*

Similar a la función de activación ReLU, pero su uso es más costoso.

$$Softplus(y) = \ln(1 + e^{y}) \tag{3.6}$$

Función Tangente Hiperbólica

Útil la hora de construir redes neuronales recurrentes.

$$\tanh(y) = \frac{e^{y} + e^{-y}}{e^{y} - e^{-y}} \tag{3.7}$$

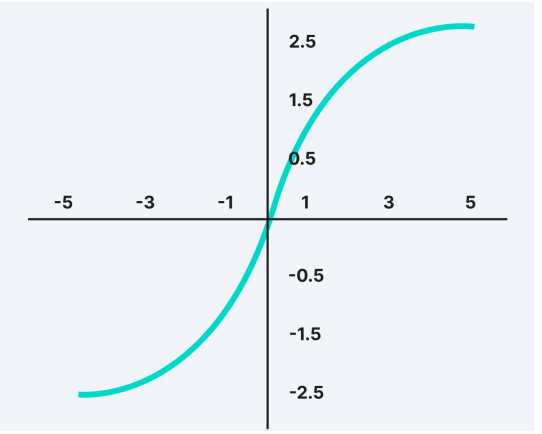


Figura 3.6: Función tanh [2]

3.3. Redes Neuronales Artificiales

Las redes neuronales artificiales son un modelo de computo formado por capas. Donde cada capa está formada por un conjunto de neuronas artificiales. Todas las capas que conforman un modelo están conectadas como un grafo acíclico dirigido. Un modelo puede estar formado por diferentes tipos de redes neuronales. La capa de entrada tiene que estar formada por el mismo número de neuronas que entradas, la última capa es la encargada de agrupar o clasificar en los grupos o clases según los patrones ocultos y las características descubiertas por el resto de capas.

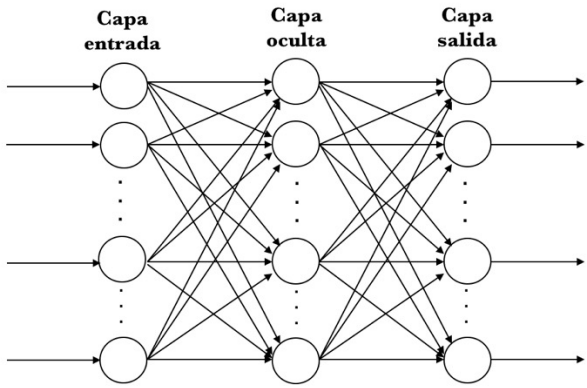


Figura 3.7: Red Neuronal Artificial [3]

Como se observa en la figura 3.7 la capa oculta y la capa de salida son lo que se denominan capas densas o *dense layers*. En capas densas, cada neurona de la capa densa esta conectada con cada neurona de la capa que la precede. El vector de salida de la capa oculta, que sirve para crear la matriz de entrada de la siguiente capa, se calcula aplicando la función de activación a la suma del bias más el producto de la matriz de pesos de las conexiones por el vector entrada (vector entrada = vector salida de la capa predecesora)

$$salida = función\_activación(producto\_matricial(Kernel, entrada) + bias) \tag{3.8}$$

El kernel en 3.8 es el vector de pesos por cada neurona de la capa predecesora. Para entenderlo mejor, en 3.9 podemos observar de que consiste la entrada, el *kernel* y el *bias*. En realidad 3.8 y 3.9 surge de aplicar como se consigue la salida de un perceptrón pero está vez hay de 1 a m perceptrones en la capa y las mismas entradas para cada perceptrón (mirar 3.1 y figura 3.2).

$$salida = activación\left(\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}\right) \tag{3.9}$$

3.3.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales o *convolutional neural networks* (CNN), son redes neuronales artificiales que suelen utilizarse para tareas de visión artificial como lo son la clasificación y segmentación de imágenes y vídeo [44]. No solo sirven para manejar imágenes y vídeos, estas redes neuronales además pueden ser útiles a la hora de clasificar textos o secuencias de datos numéricos. Si se crea un modelo que contenga capas en las que se combinen redes neuronales covolucionales con redes neuronales recurrentes.

Entre las propiedades que tienen las redes convolucionales, cabe destacar que aprenden patrones sin importar la posición de las características y aprenden jerarquías especiales de los patrones [4].

- **Aprenden patrones sin importar la posición de las características**
  Las redes neuronales convolucionales están formadas por capas convolucionales, que se diferencian de las capas densas en que las capas convolucionales aprenden patrones localmente y las capas densas aprenden globalmente. Por ejemplo, si estamos identificando números escritos a mano en una imagen. Si el numero puede aparecer en cualquier zona de la imagen, entonces mediante capas convolucionales se puede extraer un patrón local: el número que es. Si el número aparece en una misma zona se puede mediante capas densas extraer un patrón global: si la imagen tiene un número en esa posición.

- **Aprenden jerarquía especiales de los patrones**
  Si mediante una capa convolucional obtienes patrones, con una segunda capa que la proceda se obtendrán patrones más generales de los patrones obtenidos en la capa predecesora.

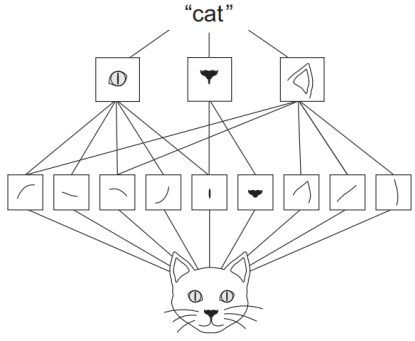


Figura 3.8: Jerarquía de patrones [4]

Las neuronas de una capa convolucional suelen utilizar la función de activación ReLU, esto se debe a la forma en que se calcula cada convolución que explicaré a continuación.

¿Qué es una convolución?

Una convolución es el proceso de aplicar el mismo filtro o *kernel* sucesivamente a de los datos de entrada, utilizando el filtro para conseguir un mapa de características.

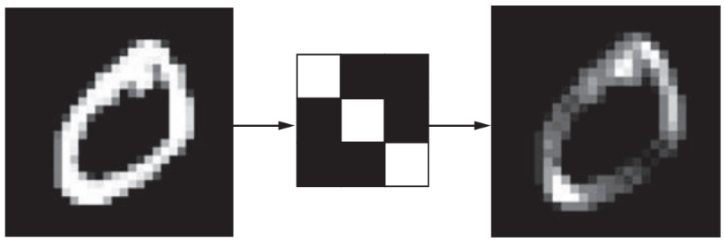


Figura 3.9: Convolución visual [4]

Un mapa de características muestra la presencia de patrones en diferentes lugares de los datos de entrada. En la figura inferior podemos observar como se halla el mapa de características usando un filtro sucesivamente a los datos preprocesados.

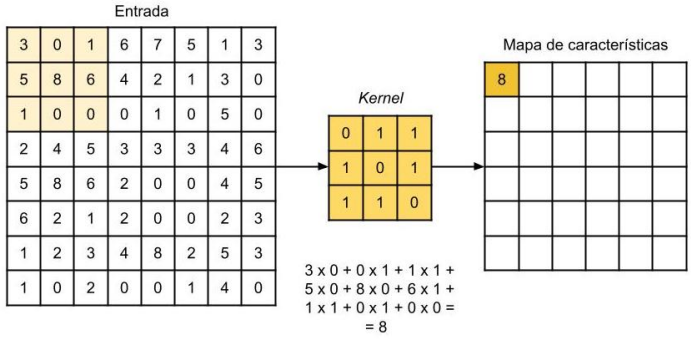


Figura 3.10: Convolución real

Si la entrada de datos es un vector, matriz o matriz-multidimensional el filtro seguirá la misma estructura de datos.

Strides y *Padding*

El *padding* consiste en añadir un determinado numero de filas y columnas a cada lado de la entrada de datos y los *strides* o pasos es la distacía que existe entre convoluciones sucesivas en los datos de entrada.

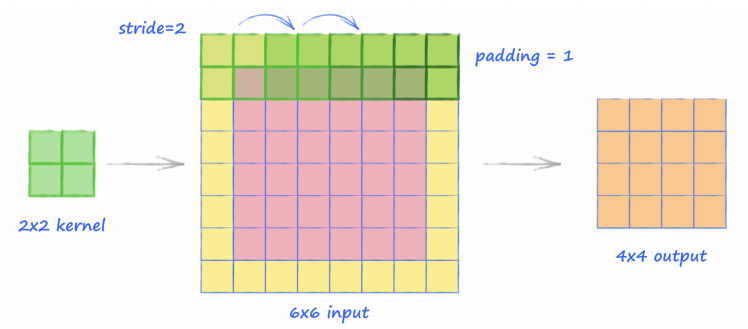


Figura 3.11: Convolución con stride y padding [5]

Capas *Pooling*

Estas capas se colocan generalmente después de una capa convolucional y sirven para reducir el tamaño del mapa de características de forma que halla una mínima pérdida de información, haciendo que la siguiente capa no tenga tantos datos de entrada. Hay dos tipos de capas de reducción o *pooling*.

- ***Max Pooling*:** Se dividen los datos de entrada en zonas de un determinado tamaño de *pool*, y se halla los máximos de esas zonas.
- ***Average Pooling*:** Como la capa *max pooling* pero en vez de hallar el maximo, se halla la media de cada zona.

3.3.2. Redes Neuronales Recurrentes

Las redes neuronales recurrentes o *recurrent neural networks* (RNN), son redes neuronales artificiales que suelen utilizarse para tareas en las que hay que tratar secuencias de datos. La propiedad característica de las redes neuronales es que son capaces de retener información.

Capa Recurrente Simple

Cada entrada en la lista de entradas en una red recurrente tiene que tener cierta dependencia o cierta asociación con la entrada anterior y posterior.

Para entender mejor como retiene información una red recurrente, explicaré como se obtiene las salidas de una red recurrente a partir de las entradas. Para ello, por cada entrada se obtiene una salida aplicando 3.10 y se guarda el estado de la salida anterior

$$salida = activación(prod(K, entrada) + prod(R, salida\_anterior) + b) \tag{3.10}$$

En la función 3.10 para hallar la salida se calcula aplicando una función de activación, que suele ser la tanh, a la suma del bias más producto matricial de la matriz de pesos del *kernel* por el vector de los datos de una entrada, hay que añadirle además otro producto matricial de la matriz de pesos del los estados por el vector de los datos de la salida anterior.

La dimensión de salida viene determinada por las dimensiones de *K, entrada, R* y *salida_anterior*. K es de dimensión *número_de_neuronas* × *características_de_cada_entrada* y cada entrada contiene un número fijo de características. Donde la entrada es cada marca del tiempo y las características son los datos de una marca del tiempo. R es de tamaño *número_de_neuronas* × *número_de_neuronas* y salida_anterior es de tamaño *número_de_neuronas*. Con esto, el resultado del producto entre K y entrada es un vector de tamaño *número_de_neuronas*, el resultado del producto de R por salida_anterior es de tamaño *número_de_neuronas*. Al tener el mismo tamaño los tres vectores de la suma en 3.10, la suma se puede efectuar.

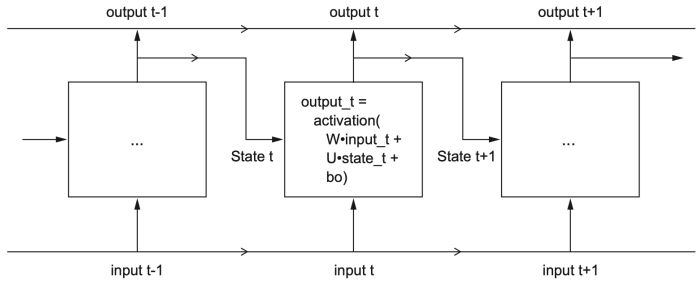


Figura 3.12: Diagrama de una capa recurrente simple [4]

También existen las capas recurrentes LSTM, *Long-Short Term Memory*, y las GRU, *Gated Recurrent Unit* que son una versión más completa y compleja de una capa recurrente.

Capa recurrente bidireccional

Las capas recurrentes pueden retener tanto información del pasado como del futuro, dependiendo de si se itera por la secuencia en orden ascendente (retiene información del pasado), como descendente (retiene información del futuro). Si hacen ambas iteraciones se retendrá información del pasado y del futuro, útil cuando la secuencia no depende del orden.

3.3.3. Activación *Softmax*

La activación *softmax* es útil cuando una entrada se puede clasificar o agrupar en más de dos clases objetivo y cuando cada entrada no puede ser clasificada o agrupada en más de una clase. Para ello calcula las probabilidades de cada clase objetivo sobre todas las clases posibles [45].

$$softmax(y_i) = \frac{e^{y_i}}{\sum_{j=1}^{m} e^{y_j}} \tag{3.11}$$

La función de activación softmax se calcula para cada neurona i, habiendo m neuronas y donde y es 3.1, el resultado de sumar el *bias* al sumatorio de los pesos por las entradas de la neurona.

3.4. Aprendizaje de las Redes Neuronales

En esta sección explicaré como se actualizan los pesos o *weights* cada época o *epoch*. Las épocas son el número de veces que se actualizan los pesos de las capas neuronales por el optimizador. Recordar que el funcionamiento del aprendizaje profundo del apartado 3.1.1 seguía estos pasos:

1. ***Forward Pass* o *Feedforward***: Las capas neuronales se ejecutan en el orden indicado a partir de las entradas obteniendo las predicciones.
2. ***Loss Function* o Función de Pérdida**: Se calcula la diferencia entre los valores esperados y las predicciones.
3. ***Backpropagation* y el Descenso del Gradiente**: La función de pérdida genera una señal de retroalimentación que mediante la propagación hacia atrás se envía la 'señal' a cada capa, luego mediante el descenso del gradiente se ajustan los parámetros internos de cada capa.

3.4.1. Función de Pérdida

También conocido por *loss function*, es la forma de evaluar si se ha predicho correctamente las predicciones obtenidas de la red neuronal.

Como con las funciones de activación, dependiendo del problema a resolver utilizaremos una función de pérdida determinada. Generalmente, para problemas de clasificación se utiliza como función de pérdida la entropía cruzada o *cross entropy* y para los problemas de regresión se suele emplear el error cuadrático medio o *mean squared error* (MSE).

$$cross\_entropy\_loss = -\sum_{c=1}^{M} y_{o,c} \cdot \log(p_{o,c}) \tag{3.12}$$

- M - número de clases [46].
- $y_{o,c}$ - 1 o 0, dependiendo de si la clasificación es correcta [46].
- $p_{o,c}$ - probabilidad predicha de la observación o de la clase c [46].

Para problemas de regresión como ya he comentado, se utiliza el error cuadratico medío, pero además se puede utilizar en algún problema el error absoluto medio o *mean absolute error* (MAE).

$$MSE = \frac{1}{m} \cdot \sum_{i=1}^{m} (y_i - \hat{y}_i)^2 \tag{3.13}$$

$$MAE = \frac{1}{m} \cdot \sum_{i=1}^{m} |y_i - \hat{y}_i| \tag{3.14}$$

- m - número de muestras.
- y_i - valor real.
- $\hat{y}_i$ - valor predicho.

La principal diferencia entre MSE y MAE es que al MAE le afectan menos los valores límite que al MSE.

3.4.2. Descenso del Gradiente

Es un método que sirve para hallar un único mínimo de una función diferenciable y convexa. El método consiste en ir encontrando puntos cada vez más cercanos al mínimo.

$$x^{(i+1)} = x^{(i)} - \alpha \cdot \frac{d}{dx} f(x) \tag{3.15}$$

- $\alpha \in (0, 1)$ - La constante tasa de entrenamiento o learning rate.

En 3.15 observamos que el nuevo punto x en el paso $i+1$ se desplaza en dirección opuesta a la derivada. Como el signo de la derivada nos indica:

- Si $f'(x) > 0 \leftrightarrow f(x)$ es creciente.
- Si $f'(x) < 0 \leftrightarrow f(x)$ es decreciente.

Si queremos localizar un mínimo el punto debe desplazarse en sentido contrario a la derivada. La derivada también regula la tasa de entrenamiento, acortando la distancia entre los puntos obtenidos en pasos consecutivos a medida que se acerca a un mínimo.

Si se generaliza para n variables obtenemos:

$$\begin{aligned} x_1^{(i+1)} &= x_1^{(i)} - \alpha \cdot \frac{d}{dx_1} f(x_1, x_2, ..., x_n) \\ x_2^{(i+1)} &= x_2^{(i)} - \alpha \cdot \frac{d}{dx_2} f(x_1, x_2, ..., x_n) \\ &\vdots \\ x_n^{(i+1)} &= x_n^{(i)} - \alpha \cdot \frac{d}{dx_n} f(x_1, x_2, ..., x_n) \end{aligned} \tag{3.16}$$

Si vectorizamos las expresiones 3.16

$$x_{(i+1)} = x_{(i)} - \alpha \cdot \nabla f(x_1, x_2, ..., x_n) \tag{3.17}$$

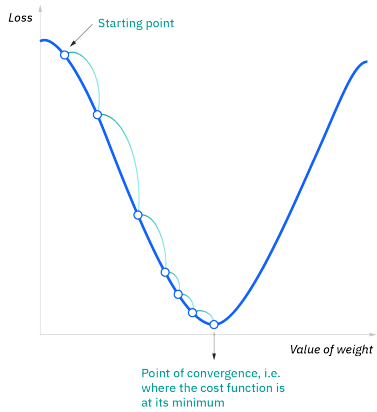


Figura 3.13: Descenso del Gradiente [6]

En la expresión 3.17 aparece el símbolo ∇, que corresponde con el gradiente. El gradiente es un vector compuesto por las derivadas parciales de una función. Una derivada parcial es la derivada de una función respecto de una variable.

$$\nabla f(x_1, x_2, ..., x_n) = \left(\frac{df}{dx_1}, \frac{df}{dx_2}, ..., \frac{df}{dx_n} \right) \tag{3.18}$$

Otros Tipos de Optimizadores

- **Descenso del Gradiente por Lotes**
  Procesa todas las muestras del conjunto de entrenamiento para actualizar los pesos de todas las capas.

- **Descenso del Gradiente Estocástico**
  Semejante al descenso del gradiente por lotes pero en vez de usar todas las muestras de entrenamiento y hacer una única actualización de los pesos de todas las capas, se utiliza cada muestra para actualizar los pesos de todas las capas tantas veces como muestras haya en el conjunto de datos de entrenamiento.

- **Descenso del Gradiente por Mini Lotes**
  Similar al descenso del gradiente estocástico, pero en vez de utilizar una única muestra de entrenamiento para actualizar los pesos de la red neuronal, se utiliza un subconjunto de las muestras de entrenamiento para hacer una actualización en los parámetros internos de la red neuronal.

- **Optimizadores más Complejos**
  Entre ellos se encuentras *AdaGrad*, *RMSProp*, *SGDNesterov*, *AdaDelta*, *Adam*. Destacar que el optimizador que mejor se adapta a todo tipo de problemas es el optimizador *Adam* [47].

3.4.3. *Feedforward* y *Backpropagation*

En este apartado explicaré como se ajustan los parámetros internos de una red neuronal.

Feedforward

En una red neuronal de L capas, donde la primera capa es la capa de entrada, cada capa genera los valores de entrada de la siguiente capa 3.19.

$$\begin{aligned} x_2 &= f_2 \equiv activación_{(2)}(W_2 \cdot x_1 + b_2) \\ &\vdots \\ x_l &= f_l \equiv activación_{(l)}(W_l \cdot x_{l-1} + b_l) \\ &\vdots \\ x_L &= f_L \equiv activación_{(L)}(W_L \cdot x_{L-1} + b_L) \end{aligned} \tag{3.19}$$

Después de obtener la salida de la última capa se computa la pérdida 3.20.

$$C \equiv pérdida(x_L, y) \tag{3.20}$$

Backpropagation

La propagación hacia atrás o *backpropagation* se encarga de propagar el error δ_L 3.21 iterativamente, empezando por la última capa L y acabando en la segunda. Recordar que la primera capa es la capa de entrada y no utiliza ni pesos ni sesgos o *bias*.

$$\delta_L = \nabla_{f_L} C \odot f'_L \tag{3.21}$$

Para entender la expresión 3.21, debemos entender las tres subexpresiones contiene:

- $\nabla_{f_L} C \rightarrow$ Vector del mismo tamaño que el número de neuronas de la última capa, en cada posición i habrá una derivada de la función de pérdida C respecto de la función de activación de la neurona i en la capa L. Expresa la tasa de cambio de la función de pérdida C respecto a las activaciones de la salida [48].
- $\odot \rightarrow$ El producto de *Hadamard* se utiliza para multiplicar el elemento i-ésimo de un vector por el elemento i-ésimo de otro vector.
- $f'_L \rightarrow$ Vector que contiene las imágenes de aplicar la derivada de la función de pérdida a todas las neuronas de la capa L. Mide como de rápido la función de activación f_L está cambiando los parámetros de la capa L [48].

Se puede hallar de manera iterativa el error de cada capa iterando de la ultima capa, hasta la segunda $i = L - 1$ .. 2.

$$\delta_l = ((W_{l+1})^T \cdot \delta_{l+1}) \odot f_l' \tag{3.22}$$

En la expresión 3.22 se puede interpretar que el error se transmite de la capa $l + 1$ a la capa l propagando el error hacia atrás por por las conexiones W con las neuronas de la capa anterior.

Ahora que ya tenemos los errores δ para todas las capas, solo nos queda actualizar los pesos y los *bias* o sesgos de cada capa. Aplicando el descenso del gradiente, se podrá actualizar los pesos y los sesgos. Para ello se itera la expresión 3.23 por $l = L$ .. 2 .

$$\begin{aligned} W_l^{new} &= W_l^{old} - \alpha \cdot (\delta_l \cdot (f_{l-1})^T) \\ b_l^{new} &= b_l^{old} - \alpha \cdot \delta_l \end{aligned} \tag{3.23}$$

El algoritmo de *backpropagation* solo computa el gradiente de la función de pérdida C para una único ejemplar de entrenamiento [48].

3.5. Técnicas Avanzadas

En esta sección explicaré algunas técnicas que pueden ser de utilidad a la hora de resolver un problema de *deep learning*. Las técnicas más importantes son la transferencia de aprendizaje, la optimización de la tasa de aprendizaje α y la regularización.

3.5.1. Transferencia de Aprendizaje

Mediante problemas similares ya resueltos, se utiliza la solución del problema parcialmente o totalmente para ayudarnos resolver nuestro problema. Para ello existen modelos pre-entrenados, la mayoría basados en redes neuronales convolucionales.

Para poder útilizar un modelo pre-entrenado, debemos conocer que un modelo de una red neuronal convolucional se divide en dos partes:

1. **Base convolucional** - Todas las capas convolucionales, teniendo en cuenta las capas *pooling*. Su principal función es extraer características.
2. **Clasificador** - Todas las capas densas utilizadas justo después de la base convolucional, para clasificar la muestra en función de las características obtenidas de la base convolucional.

A la hora de crear la red neuronal para resolver el problema, podemos seguir tres estrategias.

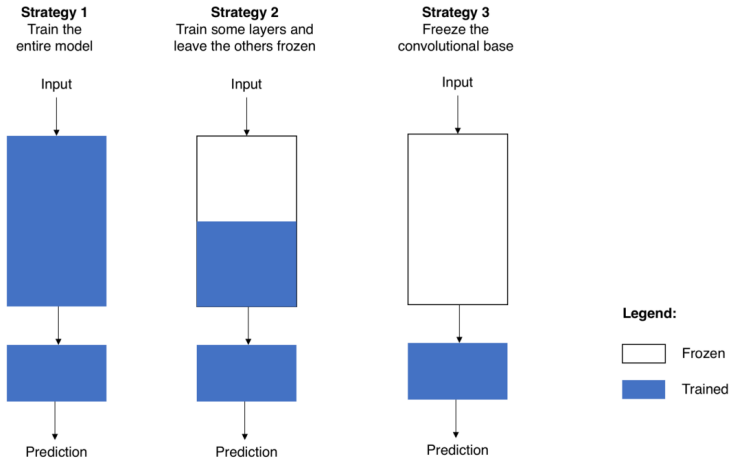


Figura 3.14: Diferentes formas de usar modelos pre-entrenados [7]

- Entrenar todo el modelo.
- Añadir nuevas capas convolucionales propias a la base convolucional pre-entrenada. Para ello se 'congelan' las capas de la base convolucional pre-entrenada para no entrenar esas capas con las nuevas muestras de entrenamiento [7].
- Usar la base convolucional, sin añadir capas convolucionales propias, cambiando sólo el clasificador. Se 'congelan' todas las capas de la base convolucional para cuando se entrene el modelo, no se modifiquen los parámetros internos de la red neuronal de las capas convolucionales ya entrenados [7].

3.5.2. Conceptos Importantes

En este apartado explicaré otros conceptos que son importantes a la hora de implementar una solución usando técnicas de *deep learning*.

Época

La época, ciclo o *epoch* el número de veces que se ejecuta el *feedforward* y el *backpropagation*. En cada época todos los datos de entrenamiento pasan por la red neuronal (*feedforward*) y esta aprende (*backpropagation*). [49].

Tasa de Aprendizaje

Hiperparámetro modificable de la red neuronal, α en 3.17, especifica el tamaño de cada paso al aplicar el descenso del gradiente. Si α es muy pequeño, puede que al aplicar el descenso del gradiente se quede atrapado en un mínimo local. Si al contrario, la tasa de aprendizaje es muy alta puede que no llegue a alcanzar el mínimo [50].

Tamaño del lote

El número de muestras que se procesan en cada iteración de un ciclo o *epoch*. Las iteraciones son el numero de lotes necesarios para completar un ciclo o *epoch* [51].

Regularización

Es un conjuto de técnicas que pueden prevenir que un modelo sufra de sobreajuste o *overfitting*, mejorando la precisión del modelo cuando procesa datos completamente nuevos. Las técnicas más utilizadas son el *dropout*, la regularización de pesos y la parada temprana [52].

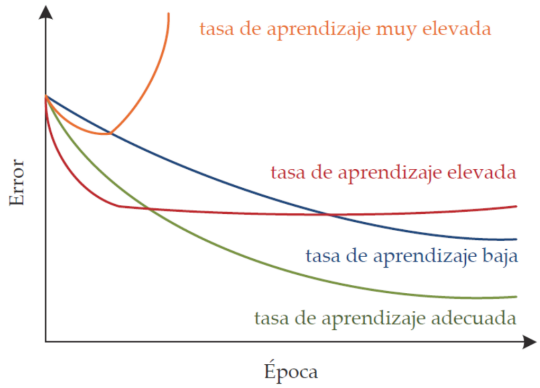


Figura 3.15: Adecuada tasa de aprendizaje [8]

- ***Dropout***
  Deshabilita aleatoriamente neuronas de la red neuronal con una cierta probabilidad que se le especifica, para ello se sustituyen las salidas de las neuronas seleccionadas aleatoriamente por 0.

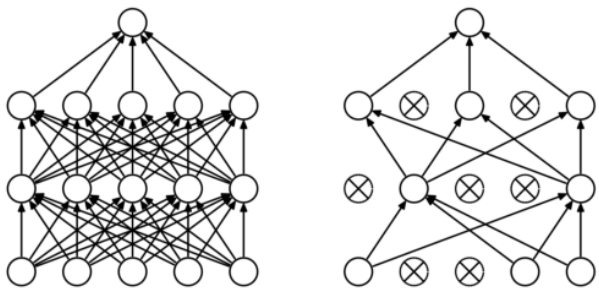


Figura 3.16: Antes y después de aplicar dropout [9]

- **Regularización de los Pesos**
  Similar al dropout, pero en vez de afectar a las salidas de las funciones de activación de las neuronas, afectan a los pesos W. Existen tres tipos de regularización para los pesos.

  - **Lasso o L1**, se utiliza cuando sospechemos que varios atributos o *features* de entrada de cada muestra sean irrelevantes [53]. Fomenta que algunos pesos acaben valiendo 0.
  - **Ridge o L2**, se utiliza cuando sospechemos que varios atributos de entrada de cada muestra tengan cierta dependencia [53]. Fomenta que los coeficientes acaben siendo más pequeños.
  - **ElasticNet o L1_L2**, combina Lasso con Ridge. Utilizado cuando existen atributos irrelevantes y atributos que dependan unos de otros.

- **Parada Temprana**
  También llamado *early stopping*, se deja de entrenar el modelo cuando deja de mejorar con los datos de validación, previniendo así el sobreajuste o *overfitting*.

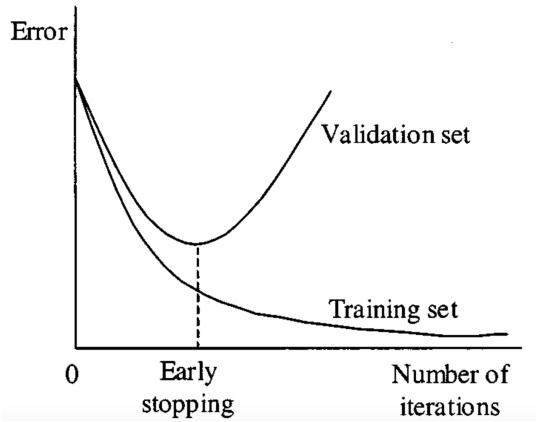


Figura 3.17: Parada temprana del entrenamiento de la red neuronal [10]

Capítulo 4

Requisitos y Planificación

Explicaré la metodología que se ha utilizado y porque se ha utilizado, además de un análisis de requisitos y riesgos. Se explicará la planificación seguida y el estudio del presupuesto de este proyecto.

4.1. Desarrollo en Cascada

Este proyecto se ha desarrollado siguiendo el ciclo de vida en *Cascada* o *One-Shot*. El modelo en *Cascada* es una metodología del desarrollo de software que ordena las fases o etapas de forma secuencial de manera que el inicio de una etapa/fase debe esperar a la finalización de la anterior.

Se decidió optar por usar el desarrollo en *Cascada* por tener los requisitos bien definidos, que no pueden cambiar a lo largo del proyecto, y una baja complejidad. El modelo en *Cascada* tiene la ventaja de que está orientado a la documentación, un aspecto importante en un Trabajo Fin de Grado.

No se decidió seguir una metodología ágil por varias razones. La primera, consiste en este proyecto se valora la documentación. Otra razón es la preferencia a seguir el plan, gracias a unos requisitos bien definidos y que no cambian a lo largo de la vida del proyecto. Además no existe la necesidad de que el cliente intervenga en el proceso, debido a que el producto solo estará disponible al finalizar el proyecto. Todas estás razones contradicen el Manifiesto Ágil.

En este trabajo existe una monitorización por parte del tutor, lo que no implica que al final de cada fase o etapa el cliente también obtenga una retroalimentación del producto que se esta desarrollando y que este pueda sugerir o exigir cambios en el producto. Con esto quiero llegar a que la monitorización del proyecto no debería importarle al cliente siempre que se cumplan el plazo de entrega del producto estipulado por el cliente.

A continuación se definirán los productos o artefactos creados en cada fase o etapa, y cuales son esas fases o etapas del desarrollo en *Cascada*.

4.1.1. Artefactos

Se crearán dos tipos de artefactos, productos o entregables durante el proyecto:

- **Productos Entregables**

  Artefactos que serán entregados al cliente al final del proyecto.

- **Productos Intermedios**

  Artefactos creados durante el proceso de creación de los entregables, cada fase o etapa generará uno de estos artefactos.

4.1.2. Fases

El proyecto consistirá de 5 fases o etapas, generando productos intermedios cada una.

1. **Planificación**

   Tiene como objetivos principales especificar las actividades y productos que se llevarán a cabo para completar el proyecto, la estimación de esfuerzo de dichas actividades, la motivación por parte mía al tener plazos en las actividades, el calculo del presupuesto del proyecto y la coordinación con mi tutor.

2. **Análisis de Requisitos**

   Se especificará que objetivos debe cubrir el proyecto, habiendo previamente analizado las necesidades de los usuarios para los que está destinado este producto software.

3. **Análisis del Sistema**

   Estudio sobre el aprendizaje automático y el aprendizaje profundo, centrándome en las redes neuronales convolucionales y redes neuronales recurrentes. Parte principal de este proyecto, ya que en esta fase se pretende encontrar la forma de resolver el problema encontrado. El estudio conlleva la lectura del libro Deep Learning with Python [4].

4. **Implementación**

   Se creará el producto entregable a partir de los conocimientos adquiridos en la fase de análisis. En esta fase se obtendrá si existe una relación entre los movimientos de una manilla de una puerta con la persona que ha abierto esa puerta.

5. **Pruebas**

   Se comprueba que los entregables creados previamente cumplan los requisitos estipulados, en el caso del *Aprendizaje Profundo* comprobar que el algoritmo utilizado funciona correctamente para datos nuevos, no con los que no se ha entrenado.

4.2. Requisitos

Los requisitos obtenidos en la fase de Análisis de Requisitos se han extraído gracias a las categorías de la tabla 2 de la página 6 de un artículo [54] sobre el análisis de requisitos en sistemas que utilizan la computación en la nube:

Requisito 1 - Funcionalidad y Fiabilidad - Identificación de un usuario
El sistema debe permitir identificar a un usuario por la información obtenida utilizando sensores en una manilla de una puerta, con una precisión mayor al 70 %.

Requisito 2 - Seguridad - Control de versiones
El sistema debe tener copias de seguridad de versiones estables, para prevenir posibles riesgos.

Requisito 3 - Reusabilidad - Componente
El sistema deberá permitir ser utilizado por otros sistemas, como un componente de una sistema mayor.

Requisito 4 - Reusabilidad - Diseño final
El sistema deberá tener un diseño reutilizable que se pueda tomar como referencia para otros sistemas similares.

Requisito 5 - Portabilidad - Cuaderno júpiter
El sistema deberá ser escrito en un cuaderno júpiter o *jupyter notebook* por ser utilizado por muchos científicos de los datos.

Requisito 6 - Facilidad del mantenimiento - Código entendible
El sistema deberá ser entendible por el 90 % de científicos de los datos.

Requisito 7 - Compatibilidad - Conexión a la red
El sistema deberá ser compatible con cualquier equipo que cuente con conexión a internet.

Requisito 8 - Eficiencia - Uso Máximo de 4GB de RAM
El sistema debe usar como máximo 4GB de RAM ya sea de la memoria principal como la memoria de la tarjeta gráfica.

Requisito 9 - Restricciones de Coste - Ejecutable gratuitamente
El sistema deberá ser ejecutado en servicio en la nube gratuito y accesible si se dispone de una conexión a internet como lo es Colab.

Requisito 10 - Tecnología - Lenguaje de programación *python*
El sistema deberá utilizar como lenguaje de programación principal *python*.

Requisito 11 - Tecnología - *Tensorflow*
El sistema deberá utilizar la librería *Tensorflow*.

Requisito 12 - Tecnología - *Keras*
El sistema deberá utilizar la librería *Keras*.

Requisito 13 - Tecnología - CPU o GPU
El sistema deberá no depender de si es ejecutado en la CPU o en la GPU.

4.3. Riesgos

A lo largo de la vida del proyecto, he estado recolectando los riesgos que afectaban al proyecto. Los riesgos estarán compuestos de una descripción, la categoría a la que pertenece, la probabilidad, el impacto, las acciones de mitigación, las acciones de contingencia, la estrategia utilizada, el alcance y el responsable.

La categoría a la que pertenece un riesgo la podemos clasificar [55] como técnica, externa, organizativa o relativa a la gestión del proyecto.

La probabilidad y el impacto se medirá según la matriz de la figura 4.1

		IMPACTO		
		baja	medio	alta
PROBABILIDAD	bajo	bajo	bajo	medio
	medio	bajo	medio	medio
	alto	medio	medio	alta

Figura 4.1: Matriz de riesgos - Probabilidad/Impacto

La estrategia puede ser aceptar, evadir, mitigar (reducir la probabilidad de que ocurra, reducir su impacto, o ambas cosas) y transferir el riesgo.

4.3.1. Lista de Riesgos

Riesgo 1 - Incertidumbre sobre la dificultad del aprendizaje profundo

- **Descripción**
  Al no tener conocimientos sobre aprendizaje automático ni aprendizaje profundo. Existe incertidumbre sobre el tiempo en adquirir los conceptos necesarios para resolver el problema adecuadamente. Si se alarga el tiempo de aprendizaje, entonces el tiempo de implementación y pruebas se tendrá que acortar. Si vemos que esos conocimientos son demasiado complejos, afectaría a la duración de la fase de análisis del sistema.
- **Categoría**
  Amenaza técnica. técnica porque depende de la dificultad de aprendizaje y uso de la nueva tecnología.
- **Probabilidad**
  Alta.
- **Impacto**
  Alto.

- **Acciones de reducción de la probabilidad**
  - Para evitar que la duración de la fase de análisis se prolongue, una opción es hacer un estudio de viabilidad previo al inicio del proyecto.
  - Consultar a alguien que haya utilizado esta tecnología.
- **Acciones de contingencia**
  - Investigar sobre la tecnología antes de ponerse a implementar la solución.
  - Pedir ayuda al tutor sobre la tecnología que se va a usar.
- **Estrategia**
  Se ha optado por reducir el riesgo. Para ello, se hará una estudio de viabilidad y si llega a materializarse se preguntará al tutor.
- **Alcance**
  Fase de análisis del sistema.
- **Responsable**
  Autor del TFG.

Riesgo 2 - Enfermedades y posibles accidentes

- **Descripción**
  Mi tutor y/o yo podemos sufrir alguna enfermedad o accidente que no nos permitiese trabajar durante algunos días, si esto ocurre la duración proyecto se vería afectada negativamente.
- **Categoría**
  Amenaza organizativa, relacionado a los recursos.
- **Probabilidad**
  Medio.
- **Impacto**
  Alto.
- **Acciones de reducción de la probabilidad**
  - Tener un margen de 2 semanas por si llega a materializarse el riesgo.
  - Tener hábitos saludables, no solo para reducir la probabilidad sino también para en caso de contraer una enfermedad o tener un accidente que este sea más leve y por lo tanto, menos negativo para la duración del proyecto.
- **Acciones de contingencia**
  - Trabajar horas extra hasta suplir las horas perdidas efecto de riesgo materializado.
  - Posponer la fecha de entrega y defensa del TFG.

- **Estrategia**
  Reducción de la probabilidad de que se produzca el riesgo, añadiendo una zona de flotación de 2 semanas a la duración total del proyecto.

- **Alcance**
  Proyecto, ya que puede darse en cualquier momento.

- **Responsable**
  Autor y tutor de este TFG.

Riesgo 3 - Perdida de los productos creados

- **Descripción**
  Si todos los productos producidos ya sean intermedios como entregables residen en un mismo dispositivo físico y este dispositivo se corrompe, perdería todo el trabajo realizado y sería necesario volver a crearlos.

- **Categoría**
  Amenaza relativa a la gestión del proyecto.

- **Probabilidad**
  Media.

- **Impacto**
  Alto.

- **Acciones de reducción de la probabilidad**
  - Utilizar sistemas en la nube, como son overleaf, google drive y git, transfiriendo el riesgo de perdida de datos.
  - Tener copias del proyecto y sus productos en distintos dispositivos.

- **Acciones de contingencia**
  - Volver a empezar el proyecto desde cero.
  - Intentar recuperar los datos de la persistencia de los dispositivos.
  - Abandonar el proyecto, si se han perdido muchos datos y no son recuperables.

- **Estrategia**
  Se ha optado por reducir la probabilidad de este riesgo usando sistemas en la nube, evitando tener que hacer copias manuales del proyecto o tener que volverlo a empezar.

- **Alcance**
  Proyecto, debido a que si se pierde un producto retrasaría el proyecto, teniendo que recuperarlo o volverlo a crear.

- **Responsable**
  Autor del TFG

Riesgo 4 - Precisión baja del modelo

- **Descripción**
  Debido a una baja precisión en el modelo, se invertirá cierto tiempo en mejorar la precisión del modelo.
- **Categoría**
  Amenaza técnica.
- **Probabilidad**
  Alta.
- **Impacto**
  Medio.
- **Acciones de reducción de la probabilidad**
  - No centrar todo el trabajo únicamente en cumplir el objetivo de tener un modelo neuronal con una alta precisión, sino hacer complementariamente una guía de usuario para usar redes neuronales.
- **Acciones de contingencia**
  - Pasar al plan B y centrarse en hacer una guía de usuario de como abordar problemas con redes neuronales, de forma más extensa que si se hubiera logrado una precisión aceptable en el modelo.
- **Estrategia**
  Se utilizará la técnica de reducción de la probabilidad del riesgo y en caso de que se materialice el riegos, se pasará a extender la guía sobre redes neuronales.
- **Alcance**
  Proyecto.
- **Responsable**
  Autor del TFG.

Riesgo 5 - Finalización de Prácticas en Empresa

- **Descripción**
  Al terminar las 300 horas de prácticas en empresa, en el caso de no volver trabajar en el mercado laboral hasta después de la obtención del título, tendría más tiempo para finalizar el TFG.
- **Categoría**
  Oportunidad Organizativa.
- **Probabilidad**
  Media.
- **Impacto**
  Medio.

- **Explotación de la oportunidad**
  - Si la empresa accede a contratarme, llegar a un consenso para que el contrato empiece después de finalizar el TFG y una vez obtenido el título del grado.
  - Se puede rechazar la oportunidad de continuar trabajando en la empresa, si se da el caso de que la empresa me ofrezca seguir trabajando. Este caso debería ser considerado solo si el TFG necesita recursos.
- **Estrategia**
  Por razones personales se optará por llegar a un consenso con la empresa. Siempre que está acceda y me proponga continuar trabajando en la empresa. Si no accede al consenso pero si me proponen continuar trabajando, la oportunidad no se materializará.
- **Alcance**
  Etapa final del proyecto, ya que el contrato de las prácticas acaba el 17 de mayo.
- **Responsable**
  Autor del TFG.

Riesgo 6 - Sin servicio temporalmente de overleaf

- **Descripción**
  Al utilizar como herramienta overleaf, un editor LaTeX online, si esta herramienta está temporalmente sin servicio, el proyecto no podría continuar hasta que vuelva a estar disponible.
- **Categoría**
  Amenaza externa.
- **Probabilidad**
  Baja.
- **Impacto**
  Alto.
- **Acciones de reducción de la probabilidad**
  - Hacer copias de seguridad locales cada semana. Previniendo así, si se prolonga la caída del servicio, poder continuar con el proyecto.
- **Acciones de contingencia**
  - Escribir, utilizando otra herramienta, todo aquello que se haya avanzado con la parte técnica. Cuando vuelva a estar disponible el servicio pasarlo al documento LaTeX de overleaf.
- **Estrategia**
  Se utilizará una estrategia de reducción, ya que el coste de realizar una copia de seguridad cada semana es ínfimo y nos previene de males mayores.
- **Alcance**
  Proyecto.

- **Responsable**
  Quienes crearon y mantienen Overleaf.

Riesgo 7 - Dificultad de comprensión del código de punto de partida

- **Descripción**
  Al partir de un código no propio, va a existir un tiempo para comprender la intención de este. Si hay puntos que no se llegan a comprenden entonces se necesitará ayuda, ya sea por parte del tutor o por parte de una investigación por mi parte.
- **Categoría**
  Amenaza técnica.
- **Probabilidad**
  Alta.
- **Impacto**
  Bajo.
- **Acciones de reducción de la probabilidad**
  - Informarse antes de como funciona las tecnologías utilizadas antes de entender el código.
- **Acciones de contingencia**
  - En caso de no entender parte del código, se pueden apuntar las dudas y preguntar al tutor en una reunión.
- **Estrategia**
  Se reducirá la probabilidad optando por informarse antes sobre las tecnologías utilizadas, por su beneficio a parte de para entender el código realizado, para modificarlo o añadir código si fuera necesario.
- **Alcance**
  Afecta al proyecto, debido a que es necesario comprender el código de partida para continuar.
- **Responsable**
  Autor del TFG.

Riesgo 8 - Incompatibilidades software/hardware para ejecutar modelos de aprendizaje profundo

- **Descripción**
  Debido a que es necesario instalar software adicional al sistema operativo, existe la probabilidad de que el software a instalar y sus dependencias sean incompatibles entre ellas o con el hardware de la maquina en la que se va a ejecutar. Esto puede provocar investigar que dependencias y hardware son compatibles o buscar alguna otra alternativa.

- **Categoría**
  Amenaza técnica.
- **Probabilidad**
  Media.
- **Impacto**
  Bajo.
- **Acciones de reducción de la probabilidad**
  - Usar un sistema en la nube que tenga el software ya instalado.
  - Buscar e instalar las versiones adecuadas de las dependencias acorde al hardware habiendo hecho una previa investigación sobre la compatibilidad del software.
- **Acciones de contingencia**
  - Al instalar software incompatible, desinstalarlo y buscar otra alternativa.
- **Estrategia**
  Se ha optado por reducir la probabilidad del riesgo, informándose antes de las compatibilidades entre las dependencias y el hardware.
- **Alcance**
  Proyecto, en concreto al paso previo a la implementación del producto software a entregar.
- **Responsable**
  Autor del TFG.

4.3.2. Prioridades de los Riesgos

Para dar prioridades a los riesgos he utilizado la tabla de la figura 4.1. Los riesgos según sus prioridades son los siguientes:

Riesgos con Alta Prioridad

1. Riesgo 1 - Incertidumbre sobre la dificultad del aprendizaje profundo.

Riesgos con Media Prioridad

1. Riesgo 2 - Enfermedades y posibles accidentes.
2. Riesgo 3 - Perdida de los productos creados.
3. Riesgo 4 - Precisión baja del modelo.
4. Riesgo 5 - Finalización de Prácticas en Empresa.

5. Riesgo 6 - Sin servicio temporalmente de overleaf.
6. Riesgo 7 - Dificultad de compresión del código de punto de partida.

Riesgos con Baja Prioridad

1. Riesgo 8 - Incompatibilidades software/hardware para ejecutar modelos de aprendizaje profundo.

4.4. Planificación

Para la planificación de este Trabajo Fin de Grado, se ha tenido en cuenta el valor de la propia asignatura: 12 créditos. Si cada crédito equivale a 25 horas de trabajo, el proyecto tendrá un esfuerzo de unas 300 horas de trabajo aproximadamente.

Para la estimación de dichas actividades se utilizará un enfoque bottom-up, que consiste en descomponer las tareas en subtareas hasta que estas subtareas puedan realizarse en una semana o dos. Después se calcula desde el esfuerzo de las subtareas, el esfuerzo de las tareas que las contienen. Así hasta obtener una estimación del proyecto.

El horario de trabajo utilizado es el siguiente. De lunes a viernes se dispondrá de 4 horas por las tardes, teniendo 1 hora más o menos en función del retraso o adelanto del proyecto real respecto a la planificación. Los domingos se trabajará 8 horas y los sábados se dedicará para la otra asignatura del grado.

A continuación explicaré como he organizado el trabajo fin de grado. Empezando desde un punto de vista más general y luego, detallando cada actividad más a fondo.

4.4.1. Planificación de Alto Nivel

En la figura 4.2 se muestra las actividades principales. A continuación explicaré lo que se va a hacer en cada actividad de la planificación desde un punto de vista general.

	Duración	Comienzo	Fin
TFG-Alfredo-Fernández-Guaza	54,25 días	mié 16/03/22	mar 31/05/22
Introducción	2,38 días	mié 16/03/22	dom 20/03/22
Planificación	10 días	dom 20/03/22	dom 03/04/22
Análisis del Sistema	20 días	dom 03/04/22	dom 01/05/22
Tecnologías Utilizadas	3,38 días	lun 02/05/22	jue 05/05/22
Implementación	16 días	jue 05/05/22	dom 29/05/22
Manuales	1,88 días	dom 29/05/22	mar 31/05/22

Figura 4.2: Planificación de alto nivel del TFG

1. Introducción: Se hará una pequeña introducción sobre este proyecto.
2. Planificación: Se creará un plan a seguir durante el proyecto.
3. Análisis del Sistema: Se hará un estudio de como se abordará el proyecto.
4. Tecnologías Utilizadas: Se explicarán las tecnología que serán necesarias para solucionar el problema planteado.

5. Implementación: Se creará el entregable producto de la resolución del problema planteado. Haciendo las pruebas necesarias para la verificación de los requisitos.
6. Manuales: Se creará la información necesaria para el uso del producto.

4.4.2. Planificación Detallada

En esta subsección explicaré en detalle cada actividad mostrado en la figura 4.2.

Introducción

1. Contexto: Se introducirá muy brevemente a lector sobre este TFG.
2. Motivación y Objetivos: Se introducirá el porqué he escogido este TFG y que es lo que se quiere conseguir con esté proyecto.
3. Estado del Arte: Se comentará otros trabajos realizados sobre aprendizaje profundo.
4. Estructura de la Memoria: Se especificará las secciones de la memoria.

	Duración	Comienzo	Fin
Introducción	**2,38 días**	**mié 16/03/22**	**dom 20/03/22**
Contexto	3 horas	mié 16/03/22	mié 16/03/22
Motivación y Objetivos	4 horas	mié 16/03/22	dom 20/03/22
Estado del Arte	4 horas	dom 20/03/22	dom 20/03/22
Estructura de la Memoria	2 horas	dom 20/03/22	dom 20/03/22

Figura 4.3: Planificación detallada - Introducción del TFG

Planificación

1. Definición del ciclo de vida utilizado: Se especificará el ciclo de vida utilizado en este proyecto y las razones por las que se ha escogido dicha metodología.
2. Planificación detallada: Se explica la planificación que se va a seguir para desarrollar este proyecto.
3. Estudio del presupuesto: Estudio sobre los costes de este TFG y comparativa entre tecnologías a utilizar.
4. Análisis de requisitos: Se identificarán especificarán los requisitos del producto entregable.
5. Gestión de Riesgos: Se identificarán los riesgos, se explicarán en detalle y se clasificarán según su prioridad.

	Duración	Comienzo	Fin
Planificación	**10 días**	**dom 20/03/22**	**dom 03/04/22**
Definición del ciclo de vida utilizado	5 horas	dom 20/03/22	lun 21/03/22
Planificación Detallada	8 horas	mar 22/03/22	mié 23/03/22
Estudio del Presupuesto	5 horas	jue 24/03/22	vie 25/03/22
Análisis de Requisitos	**1,25 días**	**vie 25/03/22**	**lun 28/03/22**
Identificación de Requisitos	5 horas	vie 25/03/22	dom 27/03/22
Especificación de Requisitos	10 horas	dom 27/03/22	lun 28/03/22
Gestión de Riesgos	**3,38 días**	**mar 29/03/22**	**dom 03/04/22**
Identificación de Riesgos	4 horas	mar 29/03/22	mar 29/03/22
Análisis de Riesgos	15 horas	mié 30/03/22	dom 03/04/22
Priorización de Riesgos	3 horas	dom 03/04/22	dom 03/04/22

Figura 4.4: Planificación detallada - Planificación del TFG

Análisis del Sistema

1. Aprendizaje Automático: Se explicarán los tipos de aprendizaje y los conceptos fundamentales del aprendizaje automático.

2. Aprendizaje Profundo: Se hará una introducción a las redes neuronales, los tipos de redes neuronales más importantes y sus carácter

	Duración	Comienzo	Fin
Análisis del Sistema	**20 días**	**dom 03/04/22**	**dom 01/05/22**
Aprendizaje Automático	**6,88 días**	**dom 03/04/22**	**mar 12/04/22**
Aprendizaje Supervisado	10 horas	dom 03/04/22	mar 05/04/22
Aprendizaje No Supervisado	10 horas	mié 06/04/22	vie 08/04/22
Aprendizaje por Refuerzo	2 horas	vie 08/04/22	vie 08/04/22
Tensores	4 horas	dom 10/04/22	dom 10/04/22
Conceptos Fundamentales	10 horas	dom 10/04/22	mar 12/04/22
Aprendizaje Profundo	**13,13 días**	**mar 12/04/22**	**dom 01/05/22**
Introducción al Aprendizaje Profundo	20 horas	mar 12/04/22	dom 17/04/22
Neurona Artificial	6 horas	dom 17/04/22	lun 18/04/22
Redes Neuronales Artificiales	**4,25 días**	**mar 19/04/22**	**lun 25/04/22**
Redes Neuronales Convolucionales	14 horas	mar 19/04/22	vie 22/04/22
Redes Neuronales Recurrentes	12 horas	vie 22/04/22	lun 25/04/22
Aprendizaje de las Redes Neuronales	16 horas	lun 25/04/22	vie 29/04/22
Técnicas Avanzadas de Aprendizaje Profundo	10 horas	vie 29/04/22	dom 01/05/22

Figura 4.5: Planificación detallada - Análisis del Sistema del TFG

Tecnologías Utilizadas

Se detallará sobre el lenguaje de programación utilizado para este proyecto y como se ha gestionado el control de las versiones del proyecto. Además de las librerías, APIs y herramientas utilizadas para llevar a cabo este proyecto.

	Duración	Comienzo	Fin
Tecnologías Utilizadas	**3,38 días**	**lun 02/05/22**	**jue 05/05/22**
Relativas a la Gestión del Proyecto	5 horas	lun 02/05/22	mar 03/05/22
Relativas a las Redes Neuronales	5 horas	mar 03/05/22	mié 04/05/22
Otras Tecnologías	5 horas	mié 04/05/22	jue 05/05/22

Figura 4.6: Planificación detallada - Tecnologías Utilizadas del TFG

Implementación

1. Crear sistema de desarrollo - Codificación: Se implementará la solución al problema, contendrá el proceso de resolución del problema.

2. Crear sistema de producción - Guía de usuario: Se guiará al usuario por la resolución final del problema, explicando la secuencia del código final.

	Duración	Comienzo	Fin
Implementación	**16 días**	**jue 05/05/22**	**dom 29/05/22**
Crear Sistema - Codificación	70 horas	jue 05/05/22	lun 23/05/22
Habilitar Sistema para Producción	20 horas	lun 23/05/22	dom 29/05/22

Figura 4.7: Planificación detallada - Implementación del TFG

Manuales

1. Manual de Usuario: Se explicará al usuario como utilizar el software creado y entender los resultados.

2. Manual de Instalación: Se especificará como instalar el software que resuelve el problema planteado.

3. Manual de Mantenimiento: Se explicará que partes del código y como se pueden modificar en caso de querer ampliar o modificar el software creado en la implementación.

	Duración	Comienzo	Fin
Manuales	**1,88 días**	**dom 29/05/22**	**mar 31/05/22**
Manual de Usuario	3 horas	dom 29/05/22	dom 29/05/22
Manual de Instalación	5 horas	dom 29/05/22	lun 30/05/22
Manual de Mantenimiento	5 horas	lun 30/05/22	mar 31/05/22

Figura 4.8: Planificación detallada - Manuales del TFG

4.5. Presupuesto

El equipo de desarrollo está formado por un ingeniero de software junior. Según el portal de empleo glassdoor [56], el sueldo promedio de un ingeniero de software junior está entorno a los 22000€ lo que equivale a 10,5€ la hora aproximadamente. El coste del ingeniero de software junior por las 300 horas de trabajo del TFG correspondería con 3150€.

En cuanto al Hardware utilizado para el proyecto. Se utilizará una RTX 2070 super valorada 529€, importante para entrenar las redes neuronales, y las demás piezas del PC están valoradas en 500€. Lo que nos da un coste del Hardware de 1029€. Según el coeficiente de amortización lineal de los equipos para procesos de información de la agencia tributaria, la amortización del PC sería de 257,25€ debido al 25 % de coeficiente lineal de amortización.

Como software, se utilizará *Anaconda, TensorFlow, Keras, Google Colab* y *Jypiter Notebooks*, al ser todos libres u *open source*, el coste es de 0€ en software.

Se ha tenido en cuenta utilizar otro entorno de desarrollo, en concreto Colab, que ofrece una plataforma en la nube para ejecutar código cuyos recursos hardware no se garantizan. Por ello, la opción gratuita de colab es perfecta para desplegar código de aprendizaje automático.

El total es la suma del coste de todos los recursos, 3150€ + 257,25€ = 3407,25€. Se añadirá a esta cifra su 5 % para tener un margen para suplir las posibles acciones de contingencia, lo que nos daría un presupuesto de 3577,61€.

4.6. Seguimiento

En este capitulo se hará una comparativa entre el tiempo estimado y el tiempo real de cada actividad. Además se describirán los cambios realizados durante el proyecto, así como las dificultades encontradas. Al final del capitulo, se hará una comparativa entre el presupuesto y el coste real.

4.6.1. Tiempo Real

En esta sección se comparará el tiempo estimado con el real, mostrando la fecha de finalización de cada tarea o conjunto de tareas.

Tarea - Planificación General	Tiempo Estimado	Tiempo Real	Fecha Fin Real
Introducción	13h	19h	22/03/22
Planificación	55h	66h	07/04/22
Análisis del Sistema	114h	138h	11/05/22
Tecnologías Utilizadas	15h	13h	15/05/22
Implementación	90h	107h	10/06/22
Manuales	13h	11h	13/06/22
Total	**300h**	**354h**	

Tabla 4.1: Resumen Planificación

Introducción

En lo que hacía la introducción sobre este proyecto, me costó definir los objetivos debido su alto impacto en el propio proyecto.

Tarea - Introducción	Tiempo Estimado	Tiempo Real	Fecha Fin Real
Contexto	3h	4h	16/03/22
Motivación y Objetivos	4h	8h	20/03/22
Estado del Arte	4h	5h	21/03/22
Estructura de la Memoria	2h	2h	22/03/22
Total	**13h**	**19h**	

Tabla 4.2: Seguimiento - Introducción

Planificación

En cuanto a la planificación, la parte más complicada fue la planificación detallada y la más tediosa la gestión de riesgos. En la planificación detallada empecé dividiendo el proyecto en introducción, planificación, análisis del sistema, tecnologías, implementación y manuales. Luego fui dividiendo cada parte en dos o más partes hasta llegar a actividades de corta duración, exceptuando la implementación por su incertidumbre.

Tarea - Planificación	Tiempo Estimado	Tiempo Real	Fecha Fin Real
Definición del Ciclo de Vida Utilizado	5h	6h	23/03/22
Planificación Detallada	8h	10h	25/03/22
Estudio del Presupuesto	5h	4h	27/03/22
Análisis de Requisitos	15h	18h	31/03/22
Gestión de Riesgos	22h	28h	07/04/22
Total	**55h**	**66h**	

Tabla 4.3: Seguimiento - Planificación

Análisis del Sistema

Durante el análisis tuve que informarme sobre el aprendizaje automático y sobre el aprendizaje profundo, cuya lectura prolongó el proyecto.

Tarea - Análisis	Tiempo Estimado	Tiempo Real	Fecha Fin Real
Aprendizaje Supervisado	10h	14h	10/04/22
Aprendizaje No Supervisado	10h	9h	12/04/22
Aprendizaje por Refuerzo	2h	1h	12/04/22
Tensores	4h	8h	14/04/22
Conceptos Fundamentales	10h	15h	18/04/22
Introducción al Aprendizaje Profundo	20h	24h	24/04/22
Neurona Artificial	6h	6h	26/04/22
Redes Neuronales Convolucionales	14h	17h	01/05/22
Redes Neuronales Recurrentes	12h	13h	03/05/22
Aprendizaje de las Redes Neuronales	16h	20h	08/05/22
Técnicas Avanzadas de Aprendizaje Profundo	10h	11h	11/05/22
Total	**114h**	**138h**	

Tabla 4.4: Seguimiento - Análisis del Sistema

Tecnologías Utilizadas

Esta parte fue la más fácil de hacer en mi opinión, nada por destacar.

Tarea - Tecnologías Utilizadas	Tiempo Estimado	Tiempo Real	Tiempo Fin Real
Relativas a la Gestión del Proyecto	5h	6h	12/05/22
Relativas a las Redes Neuronales	5h	4h	13/05/22
Otras Tecnologías	5h	3h	15/05/22
Total	**15h**	**13h**	

Tabla 4.5: Seguimiento - Tecnologías Utilizadas

Implementación

Al tener que esperar cierto tiempo por cada ejecución y al tener que analizar las métricas a la vez que aprender conceptos nuevos retrasaron el proyecto.

Tarea - Implementación	Tiempo Estimado	Tiempo Real	Tiempo Fin Real
Crear Sistema	70h	83h	05/06/22
Habilitar Sistemas para Producción	20h	24h	10/06/22
Total	**90h**	**107h**	

Tabla 4.6: Seguimiento - Implementación

Manuales

Al no existir manual de usuario, ya que su uso meramente pulsar un botón, se optó por unificar el manual de usuario con el de instalación, más tarde se realizó el manual de mantenimiento.

Tarea - Manuales	Tiempo Estimado	Tiempo Real	Tiempo Fin Real
Manual de Usuario	3h	0h	10/06/22
Manual de Instalación	5h	6h	12/06/22
Manual de Mantenimiento	5h	5h	13/06/22
Total	**13h**	**11h**	

Tabla 4.7: Seguimiento - Manuales

4.6.2. Dificultades

Al principio al no tener casi nociones sobre aprendizaje automático ni sobre aprendizaje profundo, tuve que pedir ayuda a Édgar Díez Alonso un compañero que tenía conocimientos sobre aprendizaje automático, para que me aclarase algún concepto. A la hora de la implementación, al principió fue frustrante hasta que encontré la solución al problema en el preprocesado de datos. También llegué a plantearme que el problema no se podía resolver o que el *dataset* tenía pocas muestras.

A la hora de evaluar si el modelo creado fue difícil hasta que utilicé la validación k-fold para comparar de forma más precisa entre distintos modelos.

La parte que más costó expresar fue como funciona una red neuronal matemáticamente, debido a su alta complejidad. Tuve un par de reuniones con el tutor para que observara el progreso del proyecto y ayudarme en ciertos aspectos del mismo proyecto.

4.6.3. Coste Real

Si recordamos el presupuesto que teníamos descrito en el apartado 4.5 es de 3577,61€. Si el salario es de 10,5€ por hora y se han utilizado 357 horas reales, el coste real del recurso alumno es de 10,5€/hora · 354 horas = 3717€. El coste total será de 3717€ + 257,25€ = 3974,25€.

Nos hemos pasado de presupuesto un 11 % por lo que para próximos proyectos similares se podrá tener en cuenta un 15 % de margen en vez de un 5 % a la hora de calcular el presupuesto.

Capítulo 5

Tecnologías utilizadas

En este Capítulo se presenta un resumen de las tecnologías utilizadas tanto para la gestión del proyecto, como para el desarrollo.

5.1. Gestión del Proyecto

Estas son las tecnologías utilizadas para organizar el desarrollo proyecto, explicaré que tecnologías he utilizado para la planificación del proyecto, para la comunicación entre los integrantes del proyecto, para la creación de la documentación del proyecto y para los controles de versiones utilizados.

5.1.1. Planificación y Comunicación

En este apartado explicaré las tecnologías que he usado para llevar la planificación del proyecto y que herramientas se han utilizado para la comunicación entre los dos integrantes del TFG.

Microsoft Project

Herramienta de software creada por microsoft y utilizada para administrar proyectos. Permite planificar automáticamente un proyecto por tareas con cierta jerarquía especificando la duración de cada tarea. También se pueden gestionar los recursos que se utilizan en el proyecto. Así como hacer un seguimiento del proyecto.

Figura 5.1: Microsoft Project [11]

Microsoft Teams

Plataforma de comunicación que permite hacer reuniones virtuales, además de un servicio de almacenamiento de ficheros que permite compartir datos fácilmente entre los integrantes del proyecto. Existen aplicaciones con similares características como *Skype* o *Discord* pero se ha optado por Microsoft Teams por estar dirigida para el mundo profesional.

Figura 5.2: Microsoft Teams [12]

Útil para mantener reuniones de seguimiento con el tutor sobre el TFG.

Correo Electrónico

Esta herramienta me fue útil en los primeros contactos para acordar el tema del este TFG.

5.1.2. Documentación

A la hora de elaborar la memoria de este TFG, opté por utilizar Overleaf está basado en el editor de texto Latex. Opté por estas herramientas por el buen aspecto visual que presentaban los documentos y la exuberante opciones que tiene.

Latex

LaTeX es un sistema que permite crear documentos de textos con una alta calidad tipográfica. Destaca por su versatilidad para poder escribir cualquier tipo de documento. Pudiendo escribir cualquier tipo de texto, además de fragmentos de código y formulas de forma elegante.

Overleaf

Una herramienta colaborativa basada en el editor de textos *Latex*. Overleaf es una herramienta en la nube por lo que no hace falta descargar e instalar dependencias para usar un editor *Latex*, siendo la forma más cómoda de escribir textos científicos con formulas y código.

Figura 5.3: Overleaf [13]

5.1.3. Almacenamiento de los Cuadernos *Jupyter*

Drive y github son las plataformas donde he almacenado tanto las versiones para desarrollo, como para producción. Evitando así la perdida de datos y código.

GitHub

Una plataforma que mediante el control de versiones *git* permite alojar proyectos, principalmente software, en la nube [57].

Figura 5.4: GitHub [14]

Ha sido utilizado principalmente al principio del desarrollo, cuando se ejecutaban los primeros modelos en una maquina local.

Drive

Es un servicio de alojamiento de ficheros alojado en la nube creado por Google [58]. Que permite almacenar grandes ficheros como *datasets* y permite abrir cuadernos *jupyter* redirigiéndonos a *colab*.

Figura 5.5: Google Drive [15]

Se ha utilizado para el desarrollo del mejor modelo que he conseguido. Se ha optado por utilizar *drive* en vez de otra herramienta basada en control de versiones por la integración que existe entre *colab* y *drive*.

5.2. Redes Neuronales

En esta sección describiré el lenguaje de programación utilizado, y las herramientas necesarias para poder crear el modelo, como lo son *Keras* que utiliza de *backend TensorFlow*.

5.2.1. Python

Lenguaje de alto nivel, interpretado y tiene un tipado dinámico, todo son objetos en *python*. Muy famosa para aplicaciones científicas. Muy utilizada junto con *R* para resolver problemas de aprendizaje automático. La filosofía de *python* es tener la mejor legibilidad posible del código.

Figura 5.6: Python [16]

Es un lenguaje de programación multiparadigma, soportando la programación imperativa, parcialmente la orientación a objetos y en menor medida la programación funcional [16].

5.2.2. Tensorflow

Biblioteca desarrollada por google, de código abierto utilizada para aprendizaje automático. Tensorflow puede correr en múltiples CPUs y GPUs y está disponible en la mayoría de sistemas operativos conocidos como lo son Windows, Linux, macOS, Android y iOS [17].

Figura 5.7: TensorFlow [17]

El nombre viene dado por las operaciones algébricas que pueden realizan los vectores multidimensionales. Los tensores se utilizan en el aprendizaje profundo para hacer el *feedforward* y el *backpropagation*.

5.2.3. Keras

Es una API de Redes Neuronales que he utilizado en este TFG, usando como backend la biblioteca de TensorFlow. Recomiendo utilizar la documentación si se utiliza esta API para desarrollar un modelo de redes neuronales, ya que además ofrece guías para desarrollar un modelo de redes neuronales [59].

Figura 5.8: Keras [18]

5.3. Otras Tecnologías

Mencionaré las otras herramientas que he utilizado para el desarrollo. Comenzaré con las herramientas utilizadas para desarrollar en local como son el sistema operativo utilizado, Anaconda, el *Jupiter Notebook* y *Jupiter Lab*. Y al final comentaré la plataforma utilizada para mostrar el modelo neuronal producido.

5.3.1. Linux

Un sistema operativo Unix compuesto por software libre y de código abierto, que surgió de la creación del toolkit de compilación GNU por Richard Stallman y del núcleo cuya primera versión fue creada por Linus Torvalds.

Figura 5.9: Linux [19]

Como sistema operativo se optó por linux por que la mayoría de guía están enfocadas a un sistema operativo basado en linux, concretamente ubuntu. También se opto por este sistema operativo por la experiencia que tengo con él.

5.3.2. Anaconda

Anaconda es una plataforma para la ciencia de los datos o *Data Science*. Su principal ventaja es simplificar la gestión e implementación de dependencias [60].

Figura 5.10: Anaconda [20]

5.3.3. Jupyter Notebook y Jupyter Lab

Los *Jupyter notebooks* son aplicaciones cliente-servidor que permiten editar y ejecutar tanto código (ej. python) como elementos de textuales y gráficos (ej. figuras, ecuaciones, tablas). Estos cuadernos son ejecutables vía navegador web tanto localmente como remotamente [61]. Los cuadernos *jupyter* utilizan la extensión *.ipynb*

Figura 5.11: Jupyter Project [21]

Jupiter lab es como una versión vitaminada del clásico *Jupyter notebook*, que permite abrir múltiples *jupyter notebooks* a la vez en el propio navegador. Además de abrir una terminal en la propia pestaña del navegador.

5.3.4. Google Colaboratory

Es una plataforma que permite ejecutar código *python* en el navegador. Pensado especialmente para tareas de aprendizaje automático, ofreciendo poder ejecutar código en GPUs sin coste alguno.

Figura 5.12: Google Colaboratory [22]

Se integra fácilmente con *drive* permitiendo abrir y ejecutar cualquier cuaderno de extensión *.ipynb* en un navegador. Además está integrado con *github*, siendo este un buen sitio para 'desplegar' el cuaderno *jupyter*.

Capítulo 6

Identificación de Personas

En este capítulo se resolverá el problema de identificar a personas por su forma de abrir la puerta utilizando técnicas de aprendizaje profundo. Para ello explicaré todo el proceso de resolución del problema que he seguido.

El proceso de resolución del problema implica explicar desde qué información he partido, como he progresado en la resolución del problema y la implementación de la solución final del problema, que no tiene por que ser la óptima.

6.1. Punto de Partida

En esta sección se explicará la información sobre la que he partido a la hora de implementar la solución del problema. Para resolver el problema se utilizará el código de Jesús Vegas, los datos que se utilizan en el código y un artículo científico que explica como se han extraído los datos y sus posibles usos.

6.1.1. Artículo

El artículo *Identifying users from the interaction with a door handle* [23] que forma parte de la revista científica *Pervasive and Mobile Computing* en el que colaboró mi tutor Jesús Vegas y César Llamas. Este artículo trata de proponer un nuevo método para identificar personas usando sensores en una manilla de una puerta, para ello recogieron muestras de 47 individuos y realizaron un estudio de que con que técnica de aprendizaje automático se pueden tener mejores resultados.

Un aspecto que tendremos en cuenta en el apartado 6.2.6, es el de la distinta variación entre los datos de los diferentes sensores. A la hora de generar el *dataset*, se utilizaron acelerómetros y giroscopios en tres ejes *X, Y* y *Z*.

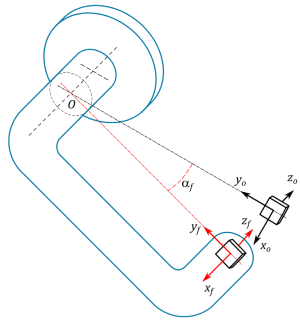


Figura 6.1: Ejes de la manilla [23]

Como la velocidad angular en el eje Z (gyrZ) y la aceleración del eje X (accX) son las que más varían, se estudiará si utilizando solo los datos de uno o dos sensores a la vez se consigue una buena precisión en el modelo. Si se logra esto, no haría falta tantos sensores a la hora de identificar a una persona por su forma de abrir la puerta.

6.1.2. *Dataset*

El conjunto de datos contiene lineas 1405300 más la cabecera. La cabecera nos explica el significado de los datos de las demás lineas. La cabecera esta formada por:

1. ***user_ID,*** identifica a que usuario pertenece la lectura de los sensores.
2. ***attempt,*** identifica la muestra de cada usuario, cada muestra está formada por 1495 medidas de los seis sensores.
3. ***time(us),*** identifica la medida de los seis sensores. Por cada intento o *attempt* las medidas vienen ordenadas crecientemente según los datos de está columna.
4. ***accX(g),*** *feature*, característica o atributo de cada medida que se obtiene por un acelerómetro en el eje de las X, mide la aceleración en el eje X usando la gravedad de la tierra como unidad de medida.
5. ***accY(g),*** *feature*, característica o atributo de cada medida que se obtiene por un acelerómetro en el eje de las Y, mide la aceleración en el eje Y usando la gravedad de la tierra como unidad de medida.
6. ***accZ(g),*** *feature*, característica o atributo de cada medida que se obtiene por un acelerómetro en el eje de las Z, mide la aceleración en el eje Z usando la gravedad de la tierra como unidad de medida.
7. ***gyrX(o/s),*** *feature*, característica o atributo de cada medida que se obtiene por un giroscopio en el eje de las X, mide la velocidad angular en el eje X usando los ángulos por segundo como unidad de medida.

8. ***gyrY(o/s),*** *feature*, característica o atributo de cada medida que se obtiene por un giroscopio en el eje de las Y, mide la velocidad angular en el eje Y usando los ángulos por segundo como unidad de medida.

9. ***gyrZ(o/s),*** *feature*, característica o atributo de cada medida que se obtiene por un giroscopio en el eje de las Y, mide la velocidad angular en el eje Y usando los ángulos por segundo como unidad de medida.

La explicación de como se han recogido estos datos se encuentra en el articulo [23] mencionado en la apartado 6.1.1.

6.1.3. Código de Partida

En este apartado mostraré el código de boceto, de extensión *.ipynb*, de partida que me entrego Jesús Vegas para que continuase yo con el proyecto. Sobretodo trata la forma preprocesar los datos para que el modelo pueda entrenarse, y para probar si el modelo generaliza correctamente.

En esta parte del código, se obtienen los datos del *dataset* en la forma adecuada para entrenar la red neuronal, teniendo un tensor 3D de datos de entrada y un vector con las etiquetas de cada muestra.

```
from tensorflow import keras
keras.__version__
# DeepKnobID

import os
data_dir = './data/'
fname = os.path.join(data_dir, 'final_dataset.csv')

f = open(fname)
raw_data = f.read()
f.close()

lines = raw_data.split('\n')
lines= lines[:len(lines)-1] # the last line is empty, take it off
header = lines[0].split(',')
lines = lines[1:]

print(header)
print(len(lines))
print("First line", lines[0])
print("Last line", lines[len(lines)-1])

# Let's convert all of these 1,405,301 lines of data into a Numpy array
import numpy as np

float_data = np.zeros((len(lines), len(header)))
for i, line in enumerate(lines):
    values = [float(x) for x in line.split(',')[:]]
    float_data[i, :] = values
```

```
30
31 # slice float_data to obtain labels
32 from keras.utils.np_utils import to_categorical
33
34 # labels, each attemp has 1495 samples, so stride is needed
35 labels = float_data[::1495, 0:1]
36 # Normalize labels from 1 - 47 to 0 - 46 range
37 labels = labels-1
38 #labels = labels.astype('int32')  # change labels to integer
39 #print(labels.dtype)
40 #print("Labels First", labels[0, ])
41 #print("Labels Last", labels[-1, ])
42
43 # Vectorize the labels
44 labels = to_categorical(labels, num_classes=47)
45
46 # Slice float_data to obtain data
47
48 # Columns in float_data np_array
49 # 0 'user_ID',
50 # 1 'attemp',
51 # 2 'time(us)',
52 # 3 'accX(g)',
53 # 4 'accY(g)',
54 # 5 'accZ(g)',
55 # 6 'gyrX(o/s)',
56 # 7 'gyrY(o/s)',
57 # 8 'gyrZ(o/s)'
58
59 #Choose the columns to include in data
60 first_column_of_data = 3
61 last_column_of_data = 9
62 number_of_columns = last_column_of_data - first_column_of_data
63 data = float_data[:,first_column_of_data:last_column_of_data]  # samples
64
65 # reshaping data to acomodate the nature of data
66
67 number_of_samples = 940
68 number_of_sensor_readings = 1495
69 data = data.reshape(number_of_samples,
70                     number_of_sensor_readings,
71                     number_of_columns) #number_of_columns)
```

Luego extrae los datos relevantes de la secuencia lecturas de sensores de una muestra y mezcla las muestras y etiquetas manteniendo la correspondencia existente entre las muestras y las etiquetas.

```
1 # extract the bath of data in each intent to be used
2 begin_of_batch = 244  # set to 0 if all, because 244 is the moment just before contact
  ↪  with the knob
3 length_of_batch = 75 #250  # 1495 if all
4 data = data[:,begin_of_batch:begin_of_batch+length_of_batch,:]  # samples
5
6 print('New shape of data tensor:', data.shape)
```

```
print("First", data[0])
print("Last", data[-1])


# Shuffle the data and labels, since samples are ordered by individual

shuffle_indices = np.arange(data.shape[0])  # create a randomized index
np.random.shuffle(shuffle_indices)
data = data[shuffle_indices]
labels = labels[shuffle_indices]
```

El siguiente fragmento de código muestra una gráfica 6.2 con las lecturas de sensores coloreado cada muestra de un color diferente.

```
from matplotlib import pyplot as plt

fig, axs = plt.subplots(figsize=(50, 27))#, layout='constrained')
for i, val in enumerate(data):
        axs.plot(data[i,:,:])
axs.grid=True
```

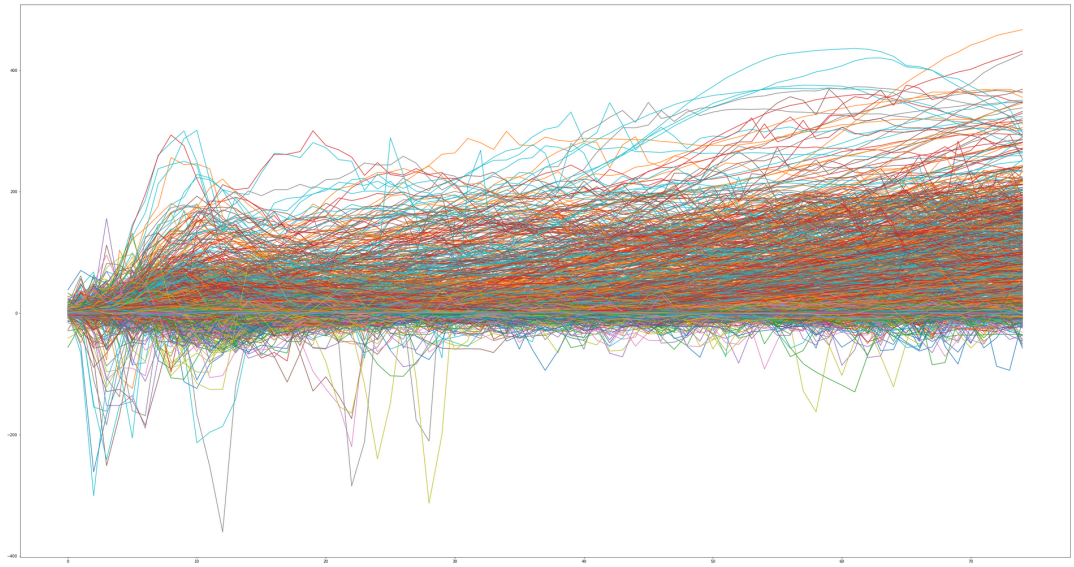


Figura 6.2: Lectura de sensores de cada muestra

Se dividen las muestras mezcladas en dos subconjuntos independientes: un subconjunto que el lo llama *train* y el subconjunto de evaluación. El subconjunto *train* esta compuesto por el subconjunto de entrenamiento y el subconjunto de validación.

```
# Define the size of the trainint, validation and test datasets
total_samples = 47 * 20  # 940
training_samples = 752   # We will be training on 80% of samples, including the 20% to
↪  validation
test_samples = 188  # We will be validating on 20% of samples

# Split the data on trainint and test sets. The validation set is
# chose by the fit method

x_train = data[:training_samples]
y_train = labels[:training_samples]

x_test = data[training_samples:]
y_test = labels[training_samples:]
```

Se normalizan los datos pasándolos a la misma escala, teniendo en cuenta que la nueva escala tiene 0 de media y 1 de desviación típica. Para ello hay que restar a los datos la media y luego dividirlo por la desviación típica.

Importante resaltar que los datos del subconjunto de evaluación se deben normalizar con la misma media y desviación típica que la utilizada para normalizar el subconjunto de entrenamiento y el de validación.

```
# Normalizing the data

# Note that the quantities used for normalizing
# the test data are computed using the training data.
# You should never use in your workflow any quantity
# computed on the test data, even for something as
# simple as data normalization. (pag. 86)

# Before normalization
BN_train_max = x_train.max()
BN_train_min = x_train.min()
BN_train_mean = x_train.mean()
BN_train_std=x_train.std()

# Normalize train data
x_train[:, :, ] -= BN_train_mean
x_train[:, :, ] /= BN_train_std

# After normalization
AN_train_max = x_train.max()
AN_train_min = x_train.min()
AN_train_mean = x_train.mean()
AN_train_std = x_train.std()

# Normalize test data with train data mean and std
x_test[:, :, ] -= BN_train_mean
x_test[:, :, ] /= BN_train_std
```

6.2. Progreso

En esta sección explicaré como a partir de los datos explicados en la sección 6.1 he conseguido dar una solución final. Después de un análisis previo del problema, y diseño de como se va a organizar el código empecé por reescribir parte del preprocesado de los datos. Tras la reescritura de preprocesado de datos, creé el modelo observando el valor devuelto por la función de perdida y la precisión del modelo con los datos de validación. Después evalué el modelo con ciertas métricas con los datos de evaluación, datos con los que no ha sido entrenado el modelo.

6.2.1. Análisis

Siguiendo los pasos de la sección 2.5, analizaré el problema primero obteniendo los datos de entrada y datos de salida del problema, luego especificaré que tipo de problema es y decidiré una forma de evaluar el modelo.

Datos de Entrada y Salida

Los datos de entrada del modelo son las muestras de diferentes usuarios o clases, y la salida es la clase o usuario a la que pertenece cada usuario. Para entrenar el modelo, este recibirá de entrada tres pares de tensores (datos, etiquetas), uno para el entrenamiento, otro para la validación y el último para la evaluación.

- **Datos:** Formado por un tensor con forma o *shape* (muestras, lecuras_de_sensores, características)
  - **Muestras:** el número de intentos, una muestra es la secuencia de lecturas de sensores de un usuario para abrir la puerta.
  - **Lecturas de sensores:** la secuencia de una muestra o intento, cada lectura de sensores está compuesta por las medidas tomadas en cierto tiempo.
  - **Características:** número de medidas tomadas por distinto sensores, cada medida de un sensor en un cierto tiempo.
- **Etiquetas:** Formado por un tensor con forma o *shape* (muestras, numero_de_usuarios), debido a la codificación *One-Hot* por ser las muestras datos categóricos.

Los datos de salida dependen de si estás usando el modelo para el desarrollo o si lo estás usando en producción.

- **Desarrollo:** Devuelve las medida de la función de pérdida o *loss* y la precisión del modelo.
- **Producción:** Mediante el método *predict*, que devuelve las clases o usuarios a los que pertenecen las muestras pasadas al mismo método *predict* del modelo.

Tipo de Problema

El problema a resolver es un problema de clasificación, ya que se tiene un *dataset* con muestras correctamente etiquetadas. La razón por la que se ha resuelto el problema mediante técnicas de aprendizaje profundo está en el artículo de partida [23].

Decisión de Evaluación del Modelo

Para evaluar el modelo se a optado por usar al principio la técnica de validación *Hold-out* debido a la rapidez con la que se obtienen las diferentes métricas. Y al final cuando se hallan obtenido modelos con buena precisión se usará la validación *K-fold* iterativa con mezcla para obtener que modelo se comporta mejor con nuevos datos (que modelo generaliza mejor).

También se opta por usar una matriz de confusión, que muestra si cada muestra del subconjunto de evaluación ha sido identificada correctamente con el usuario que hizo ese intento o muestra. También muestra cuando ha sido reconocido el intento por otro usuario, además de mostrar el usuario al que se ha reconocido erróneamente y el usuario al que debía haber reconocido.

6.2.2. Diseño

Se ha optado por dividir en secciones de código cohesivas explicando encima de cada sección que es lo que hace dicha sección de código. Dividiré el archivo *.ipynb* en 4 secciónes:

1. *Dataset,* se procesan los datos para pode entrenar con ellos el modelo.
2. *Machine learning aproach,* creación del modelo.
3. *Evaluation on test data,* evaluación *hold-out* del modelo, con la precisión y el *loss* además de la metricas obtenidas de la matriz de confusión.
4. *Iterative k-fold evaluation,* se evalúa la precisión final del modelo.

6.2.3. Preprocesado de los Datos

La mayor parte del preprocesado está hecha por Jesús Vegas, pero al mi parecer había una errata. Al aletorizar las muestras y coger un numero de muestras, a lo mejor en el subconjunto de entrenamiento y en el subconjunto de validación no hay ninguna muestra de un usuario y en el subconjunto de evaluación estarían todas las muestras de ese usuario. Por lo que el modelo no sabría como enfrentarse a cualquier muestra del mismo usuario. Para resolver este problema he aleatorizado las muestras de otra forma.

Aleatorizar las Muestras

La siguiente sección de código muestra como he aleatorizado las muestras para que los tres subconjuntos tengan muestras de las distintas clases de forma balanceada.

El método *getIndices* devuelve los índices de las muestras mezcladas y separadas en los subconjuntos de entrenamiento, validación y evaluación. Para conseguir los índices, itero por las categorías (mezcladas o no) y luego itero por las muestras (mezcladas o no) y añado a los conjuntos los índices correspondientes a la categoría y la muestra actual.

```
def getIndices(attempts=20, categories=47, training_probability=0.6,
↪  validation_probability=0.2, shuffle=True):

    training_SPC = attempts * int(training_probability * 100) // 100
    validation_SPC = attempts * int(validation_probability * 100) // 100
    test_SPC = training_SPC + validation_SPC

    train_indices = []
    validation_indices = []
    test_indices = []

    random_indices = [i for i in range(attempts)]

    shuffled_categories = [i for i in range(categories)]
    if shuffle: random.shuffle(shuffled_categories)

    for category in shuffled_categories:
        if shuffle: random.shuffle(random_indices)
        for i in random_indices[:training_SPC]:
            train_indices.append(attempts * category + i)
        for i in random_indices[training_SPC:test_SPC]:
            validation_indices.append(attempts * category + i)
        for i in random_indices[test_SPC:]:
            test_indices.append(attempts * category + i)

    return (train_indices, validation_indices, test_indices)
```

Separar los Datos en Subconjuntos

Obtengo los conjuntos a través de los índices obtenidos de en apartado anterior y de los datos y etiquetas obtenidos del *dataset* por Jesús Vegas.

```
x_train = data[train_indices,:,:]
y_train = labels[train_indices,:]

x_validate = data[validate_indices,:,:]
y_validate = labels[validate_indices,:]

x_test = data[test_indices,:,:]
y_test = labels[test_indices,:]
```

Utilizar más Lecturas por Muestra

A la hora de coger las lecturas pasé de coger 75 lecturas a coger todas salvo las primeras 244 lecturas, por no tener valor debido a ser el momento previo al contacto con la manilla. Comparar la siguiente imagen con 6.2. Esta elección se debe a que el modelo se comparte mejor cuanto más datos tenga.

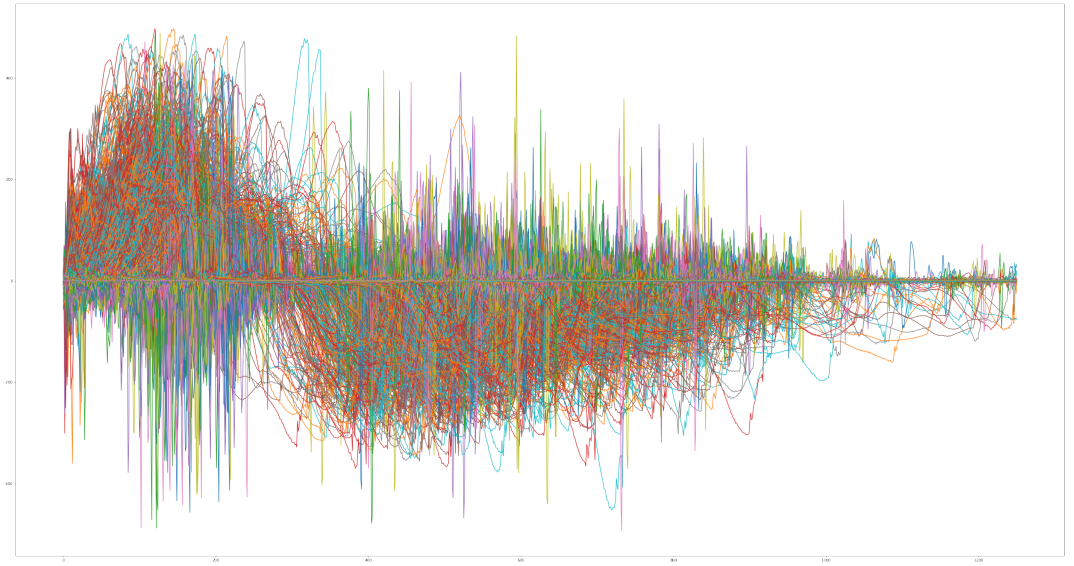


Figura 6.3: Todas las lecturas de los sensores por cada muestra

6.2.4. Creación del Modelo

En esta sección se explicará como se ha llegado al modelo neuronal final utilizando la API de *Keras*. Lo primero que he hecho es sustituir la forma en que normalizan los datos, en vez de hacerlo como Jesús Vegas, opté por usar una capa de preprocesado de Keras que realiza la normalización automáticamente. Para usar la capa *Normalization* no hay que hacer más que crearla y mediante el método *adapt* indicarle de donde queremos que obtenga la media y la desviación típica para poder normalizar cualquier dato que le demos al modelo, haciendo el modelo más portable.

Capas Neuronales Necesarias

Para resolver el problema, primero analicé que tipos de capas neuronales eran necesarias para elaborar el modelo. El tipo de capa más necesaria para este problema es una capa recurrente debido a que se analizan secuencias temporales. Otro tipo de capa neuronal no tan necesaria, es una capa convolucional. Al utilizar una capa convolucional para una secuencia de datos, podemos extraer características de las secuencias.

Si miramos en la documentación de keras, que lo recomiendo, veremos que en la sección de capas recurrentes aparecen un montón de capas recurrentes. Destacar que las capas LSTM y GRU son de las más utilizadas para este tipo de problemas, por lo que se ha optado por utilizar LTSM.

Por otra parte, las capas convolucionales que usaremos son las capas Conv1D, MaxPooling1D y GlobalMaxPooling1D. Los argumentos que aceptan las diferentes capas están bien documentados en la documentación de keras. Pero voy a explicar la forma más habitual de utilizar las diferentes capas.

- ***LTSM(units, return_sequences)*** - Las *units* son el número de características diferentes que se aprenden de la secuencia. El segundo argumento, se le pasa *True* cuando la capa siguiente necesita de entrada un tensor 3D. También tiene el campo de activación, pero en las capas recurrentes es mejor no cambiar la activación que viene por defecto.
- ***Conv1D(filters, kernel_size, activation)*** - Aplica las operaciones convolucionales a la secuencia de características. Obtiene tantos mapas de características como *filters* al aplicar convoluciones con una ventana de tamaño *kernel_size*. Como activación recomiendo utilizar ReLU.
- ***MaxPooling1D(pool_size)*** - Se suele utilizar después de utilizar una capa *Conv1D*, reduce cada mapa de características tantas veces como indiqué el pool_size *pool_size*. La capa MaxPooling1D divide los mapa de características en partes iguales de tamaño pool_size y por cada parte obtiene el máximo.
- ***GlobalMaxPooling1D()*** - Obtiene los máximos sobre la segunda dimensión, que en nuestro caso es la secuencia de lecturas de sensores. Obteniendo un valor por secuencia.

Se necesitará una capa densa al menos que sirva como clasificador, para ello se utilizará la capa *Dense(units, activation)* - Las unidades son el número de neuronas conectadas completamente con las neuronas de la capa anterior, en nuestro caso serán 47 y la activación será *softmax* para poder clasificar en 47 grupos.

También se puede optar por técnicas de regularización como el dropout o la regularización de pesos L1 y/o L2 por cada capa.

Modelo *Overfitted*

Lo siguiente es elaborar un modelo que sufra de *overfitting* (no un modelo complejo), para ello hay que poner un número de épocas, ciclos o *epochs* alto y observar que las métricas de pérdida o *loss* y de precisión del entrenamiento mejoren a la vez que la pérdida y precisión de la validación. A la hora de compilar el modelo, recomiendo usar de optimizador *Adam* por ser tan versátil con todo tipo de problemas y dejar la tasa de aprendizaje por defecto y utilizar un tamaño de lote de 16, 32, 64 o 128. Al principió es preferible fijar el tamaño de lote y la tasa de aprendizaje.

Para saber si estamos sufriendo *overfiting o underfitting* tendremos que observar si la pérdida de validación es mayor o menor que la pérdida de entrenamiento.

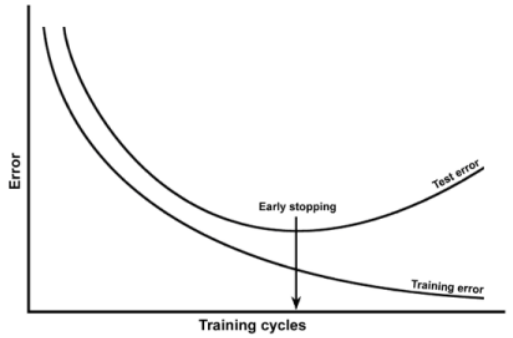


Figura 6.4: Saber si el modelo sufre de *overfitting* o *underfitting* por el *loss* [24]

Si la curva de la pérdida de validación está por encima de la curva de la pérdida de entrenamiento, el modelo se estará sufriendo de *overfitting*. Si la curva de la pérdida de validación está por encima de la curva de la perdida de entrenamiento, el modelo estará sufriendo de *underfitting*. Un modelo se dice que ajusta correctamente cuando apenas hay diferencia entre la curva de la pérdida de validación y la curva de pérdida de entrenamiento.

Tras muchas pruebas con distintos modelos, se me ocurrió usar un modelo secuencial con una primera capa recurrente seguida de una base convolucional y un clasificador utilizando en el clasificador una regularización de los pesos para combatir el overfitting.

```
normalizeLayer = Normalization(input_shape=(batch_length, columns))
normalizeLayer.adapt(x_train)

model = Sequential()

model.add(normalizeLayer)
model.add(layers.Bidirectional(layers.LSTM(16, return_sequences=True)))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(64, 5, activation='relu'))
model.add(layers.Conv1D(64, 5, activation='relu'))
model.add(layers.MaxPooling1D(3))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.Conv1D(128, 5, activation='relu'))
model.add(layers.GlobalMaxPool1D())

model.add(layers.Dense(128, activation='relu',
↪  kernel_regularizer=regularizers.l1_l2(l1=0.001, l2=0.001)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(47, activation='softmax'))
```

Modificar capas e hiper-parámetros

Para llegar al modelo final mostrado en el apartado 6.3.2 probé tres combinaciones:

- **Modelo 1,** formado por una capa recurrente bidireccional LSTM, seguida de la base convolucional y el clasificador.
- **Modelo 2,** formado por la base convolucional seguida de una capa recurrente bidireccional LSTM, y finalmente el clasificador
- **Modelo 3,** formado por una base convolucional con una capa recurrente seguida del clasificador. La capa recurrente esta situada entre capas covolucionales.

De las tres combinaciones, el modelo 3 fue el que obtuvo mejores resultados.

En mi opinión, esta es la parte más complicada ya que existen infinitas combinaciones a probar. Conseguir un modelo con buena precisión es un arte, ya que nos basamos en el método de prueba y error. Probar el modelo una y otra vez, hasta que consigamos mejorar las métricas de evaluación.

Callbacks

Los *callbacks* son objetos que pueden realizar acciones en varias etapas del entrenamiento. Puede crear un *callback* o utilizar uno ya existente. Entre los existentes que he usado yo están el *ModelCheckpoint*, y el *LearningRateScheduler*.

El *callback ModelCheckpoint* sirve para guardar el modelo entrenado automáticamente durante el entrenamiento mismo.

El *LearningRateScheduler* sirve para cambiar la tasa de aprendizaje durante el entrenamiento. Para ello hay que crear una función y pasar como parámetro al objeto *LearningRateScheduler* la función. [62]

También cree mi propio *callback* tomando como ejemplo [62]. Este *callback* hereda de la clase *Callback* y para su creación son necesarios 3 parámetros:

- ***threshold*,** probabilidad mínima que hace parar el modelo si también se cumple el *patience*.
- ***max_loss*,** si en el entrenamiento del modelo se cumple la pérdida es menor que el *max_loss* para el modelo de entrenar solo si también se cumple el *patience*
- ***patience*,** indica el número de épocas o *epochs* que deben pasar desde que empeora la métrica loss para dejar de entrenar el modelo.

```
class ThresholdCallback(Callback):
    def __init__(self, threshold, max_loss=0.1, patience=0):
        super(ThresholdCallback, self).__init__()
        self.threshold = threshold
        self.max_loss = max_loss
        self.patience = patience

        self.best_weights = None

    def on_train_begin(self, logs=None):
        self.wait = 0
        self.best = np.Inf

    def on_epoch_end(self, epoch, logs=None):
        val_accuracy = logs.get('val_categorical_accuracy')
        current = logs.get('loss')

        if np.less(current, self.best):
            self.best = current
            self.wait = 0
            self.best_weights = self.model.get_weights()
        else:
            self.wait += 1
            is_patience_end = self.wait >= self.patience
            is_out_of_threshold = val_accuracy > self.threshold
            is_less_than_max_loss = current < self.max_loss
            stop = is_patience_end and (is_out_of_threshold or is_less_than_max_loss)
            if stop:
                self.model.stop_training = True
                self.model.set_weights(self.best_weights)
```

El LearningRateScheduler que utilizo te divide la tasa de aprendizaje actual entre 2 al acabar las *epochs* o épocas que se especifiquen.

```
def scheduler(epoch, lr):
  if epoch in {15, 30, 50}:
    return lr / 2

  return lr

scheduler = LearningRateScheduler(scheduler)
```

Optimizador y Tamaño del Lote

Una vez obtenido un modelo que generalice con una buena precisión, podemos cambiar de optimizador y ver que tal se comporta con diferentes tasas de aprendizaje. Una vez tengamos el optimizador y el *learning rate* asignados, podremos variar el tamaño de lote a ver y volver a observar los resultados.

6.2.5. Evaluación del Modelo

Aparte de las típicas métricas como lo son la pérdida o *loss* y la precisión del modelo con los datos de evaluación existen otro tipo de métrica más visual es la matriz de confusión. Al final cuando tengamos dos o tres modelos definitivos, para elegir el mejor utilizaremos la técnica de k-fold iterativo.

Matriz de Confusión

La matriz de confusión es una matriz cuadrada cuyas dimensiones son de tamaño el número de clases, en este caso el número de usuarios [63]. De la matriz de confusión se pueden sacar el numero de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos. Estas métricas se pueden obtener de la matriz de confusión de la siguiente forma:

- **TP o Verdaderos positivos** del usuario i, si está en la diagonal de la matriz de confusión. Es una salida del modelo predicha en la clase correcta.
- **FP o Falsos positivos** del usuario i, la suma de la columna i-ésima quitando los TP de i. Es la salida del modelo que se predice erróneamente la clase correcta.
- **FN o Falsos negativos** del usuario i, la suma de la fila i-ésima quitando los TP de i. Es la salida del modelo que se predice erróneamente la clase incorrecta.
- **TN o Verdaderos negativos** del usuario i, son todos los valores de la matriz salvo los valores de la columna y los valores de la fila i.

Aunque sea más útil para un *dataset* con muestras de clases desbalanceadas, se puede ver visualmente los resultados con datos que no ha visto el modelo.

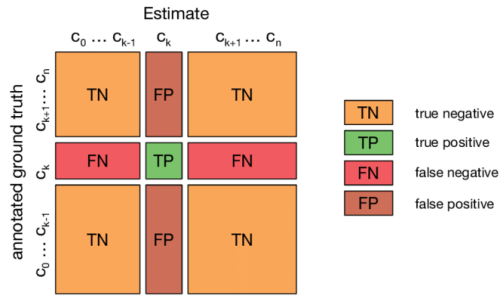


Figura 6.5: Hallar los TP,TN, FP y FN de la matriz de confusión [25]

K-fold Iterativo

Para implementar la técnica de validación k-fold iterativa, primero me he centrado en hacer solo la validación k-fold. En el método k-fold se divide el *dataset* en dos subconjutos, en el subconjunto de evaluación o *tests* y el otro subconjunto contiene los datos restantes. Tras esto, se divide el conjunto restante en k-partes iguales. Está parte la he implementado de manera similar a la mezcla de las muestras.

```
def get_folds(attempts=20, categories=47, k=4,
              tests_probability=0.2, shuffle=True):

    test_attempts = attempts * int(tests_probability * 100) // 100
    training_and_validation_attempts = attempts - test_attempts

    step = training_and_validation_attempts // k

    folds = [ [] for _ in range(k) ]
    test_indices = []

    shuffled_attempts = [i for i in range(attempts)]

    shuffled_categories = [i for i in range(categories)]
    if shuffle: random.shuffle(shuffled_categories)

    for category in shuffled_categories:
        if shuffle: random.shuffle(shuffled_attempts)
        fold_index = 0
        for j in range(0, training_and_validation_attempts, step):
          for i in shuffled_attempts[j:j+step]:
            folds[fold_index].append(attempts * category + i)
          fold_index += 1
        for i in shuffled_attempts[training_and_validation_attempts:]:
            test_indices.append(attempts * category + i)

    return (folds, test_indices)
```

El método *get_folds()* devuelve los índices de las muestras de cada *fold* y los índices de las muestras utilizadas para la evaluación. Después definí otro método llamado *k_fold()* para ejecutar la validación *k-fold* que devuelve las precisiones obtenidas en las k iteraciones de la evaluación *k-fold*.

Para que la evaluación *k-fold* sea iterativa, ejecute el método *k-fold* tantas veces como se quiera, para obtener como se comporta el modelo con datos que no había visto nunca y que la precisión no esté condicionada por las continuas modificaciones que se han realizado al modelo.

6.2.6. Comparativas

En esta sección haremos comparativas teniendo en cuenta la métrica de precisión y la métrica de pérdida para observar la diferencia entre los distintos modelos y elegir el que más convenga. Después intentaré que la red neuronal necesita las menos características o *features* posibles.

Maximizar la Precisión

En el aprendizaje automático la métrica de precisión es muy importante a la hora de evaluar un modelo. Para un modelo que clasifica en más de dos clases, la precisión se obtiene según la expresión 6.1 teniendo en cuenta que las muestras y los aciertos de las muestras son del conjunto de datos de evaluación.

$$precisión = \frac{aciertos}{muestras} \tag{6.1}$$

A continuación elegiré un modelo entre tres dependiendo de la precisión obtenida tras realizar 10 iteraciones *k-fold* con un valor de $k = 4$. Así tendremos una medida más exacta ya que se realizan 40 medidas de la precisión por cada modelo a comparar.

Todos los modelos a comparar comparten la capa de normalización y el clasificador. La base convolucional y recurrente es la que varía en los tres modelos que vamos a comparar.

```
def create_model(x_train, summary=True):

  # Preprocessing Normalization
  normalizeLayer = Normalization()
  normalizeLayer.adapt(x_train)

  input_tensor = Input(shape=x_train.shape[1:])
  common_input = normalizeLayer(input_tensor)

  # Recurrent and Covolutional Base
                .
                .
                .

  # Classifier
  output_tensor = Dense(47, activation=softmax)(tensor)

  model = Model(input_tensor, output_tensor)

  if summary: model.summary()

  return model
```

- **Modelo A**, este modelo obtiene una precisión media de 0,886 usando 93047 parámetros y su implementación es la siguiente:

```
# Recurrent and Covolutional Base
tensor = Conv1D(47, 7, activation=relu)(common_input)
tensor = MaxPooling1D(5)(tensor)
tensor = Conv1D(74, 7, activation=relu)(tensor)
tensor = MaxPooling1D(3)(tensor)
tensor = Bidirectional(LSTM(37, return_sequences=True))(tensor)
tensor = Conv1D(59, 7, activation=relu)(tensor)
tensor = MaxPooling1D(3)(tensor)
tensor = GlobalMaxPooling1D()(tensor)
```

- **Modelo B**, este modelo obtiene una precisión media de 0,869 usando 68555 parámetros y su implementación es la siguiente:

```
# Recurrent and Covolutional Base
tensor = Conv1D(47, 7, activation=relu)(common_input)
tensor = MaxPooling1D(5)(tensor)
tensor = Conv1D(74, 7, activation=relu)(tensor)
tensor = MaxPooling1D(4)(tensor)
tensor = Bidirectional(LSTM(27, dropout=0.5, recurrent_dropout=0.2,
↪  return_sequences=True))(tensor)
tensor = Conv1D(47, 7, activation=relu)(tensor)
tensor = MaxPooling1D(3)(tensor)
tensor = GlobalMaxPooling1D()(tensor)
```

- **Modelo C**, este modelo obtiene una precisión media de 0,87 usando 38391 parámetros y su implementación es la siguiente:

```
# Recurrent and Covolutional Base
tensor = Conv1D(47, 7, activation=relu)(common_input)
tensor = MaxPooling1D(5)(tensor)
tensor = Conv1D(57, 7, activation=relu)(tensor)
tensor = MaxPooling1D(3)(tensor)
tensor = Bidirectional(LSTM(24, dropout=0.2, return_sequences=True))(tensor)
tensor = GlobalMaxPooling1D()(tensor)
```

A continuación se presentará una gráfica comparativa con diagramas de caja y bigotes entre los tres modelos. Después se mostrarán las medidas tomadas en una tabla comparativa para analizar los resultados de los distintos modelos.

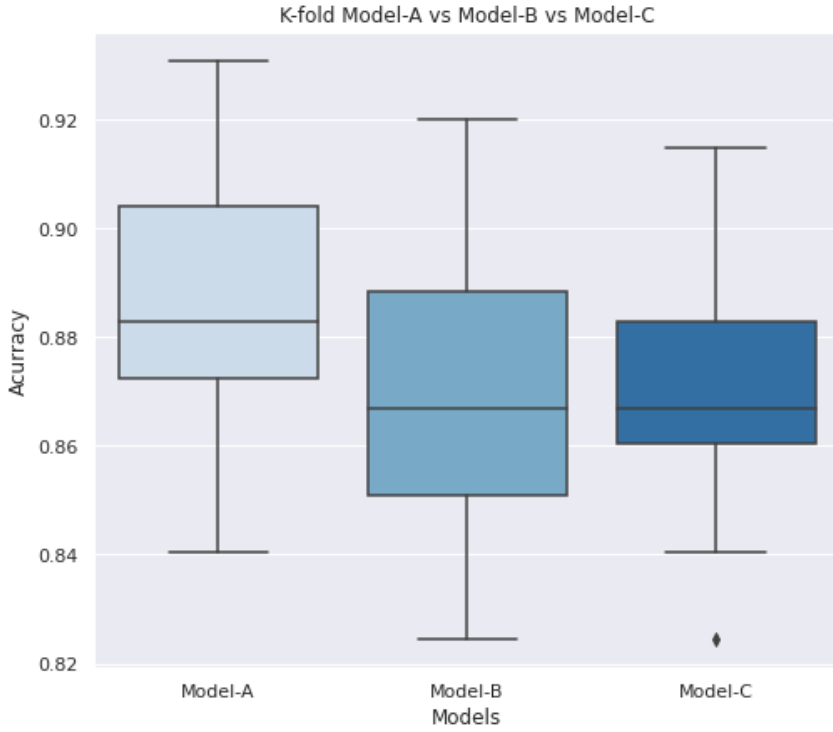


Figura 6.6: Gráfica comparativa entre modelos

	Model-A	**Model-B**	**Model-C**
media	0.886037	0.869149	0.870479
std	0.020650	0.022912	0.019180
min	0.840426	0.824468	0.824468
25 %	0.872341	0.851064	0.860372
50 %	0.882979	0.867021	0.867021
75 %	0.904255	0.888298	0.882979
max	0.930851	0.920213	0.914894

Tabla 6.1: Tabla comparativa entre modelos

Observando la tabla 6.1 podemos descartar el Modelo B ya que se comporta peor en todas las medidas que los otros dos modelos. Si comparamos el modelo A y el modelo C, vemos que ligeramente los resultados favorecen al modelo A. Sin embargo hay que tener en cuenta que el modelo C es la mitad de complejo que el modelo A.

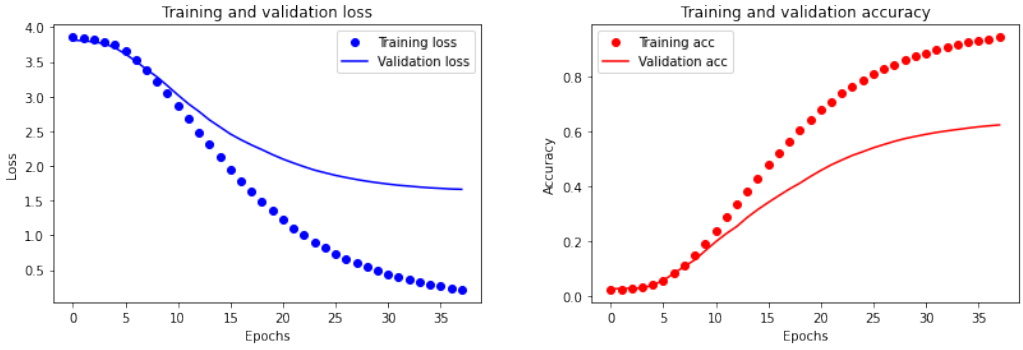
Podemos obtener dos conclusiones de la tabla 6.1. La primera conclusión se obtiene de observar que el modelo A obtiene los mejores resultados y es menos propenso a los valores límite. Y como segunda conclusión se puede observar que el modelo C es menos complejo pero ligeramente peor.

Minimizar Características

En este apartado se estudiará si es posible reducir el número de características, evitando así en una implementación real usar tantos sensores. Utilizando el modelo A del apartado anterior probaremos a utilizar solo ciertas características obtenidas por determinados sensores.

He estudiado utilizando solo los datos obtenidos por las distintas combinaciones del acelerómetro X y del giroscopio Z. Usaré las gráficas obtenidas de analizar la pérdida y la precisión entre los datos de entrenamiento y los de validación.

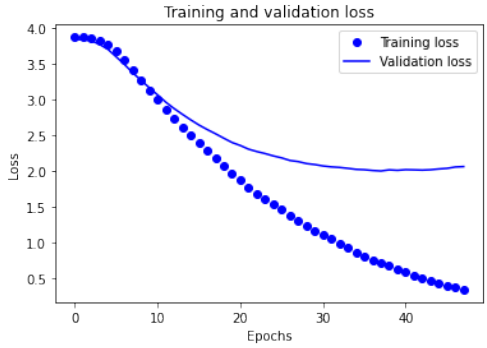
Observando las gráficas obtenidas podemos ver que el modelo sufre de *underfitting*, utilizando las técnicas para combatir el sobreajuste y el subajuste del apartado 2.5.4 llegue a la conclusión que cuanto más sensores se utilicen (más datos) mejor funcionará el modelo porque probando a modificar la tasa de aprendizaje y la complejidad del modelo no mejoraban las métricas.



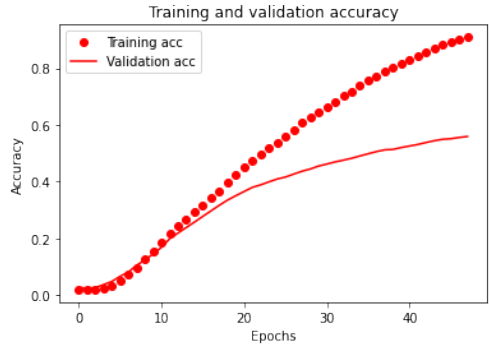
(a) Pérdida

(b) Precisión

Figura 6.7: Acelerómetro X

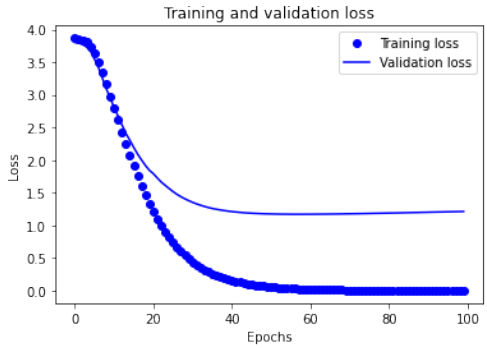


(a) Pérdida

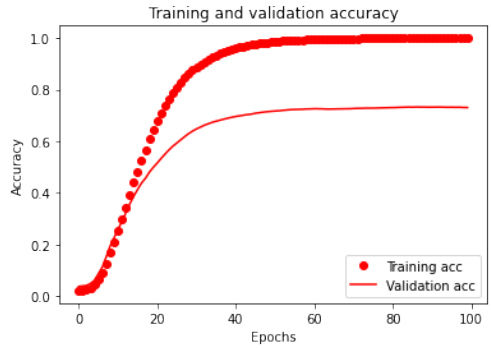


(b) Precisión

Figura 6.8: Giroscopio Z



(a) Pérdida



(b) Precisión

Figura 6.9: Acelerómetro X y giroscopio Z

6.3. Solución Final

En está sección resumiré las propiedades claves del modelo que elegido que mejor soluciona el problema de la identificación de personas a partir de datos obtenidos por sensores. Como modelo final he elegido el modelo A, comparado en la sección previa.

6.3.1. Datos Elegidos

Se ha optado por utilizar el máximo número de datos posibles por la sensibilidad del modelo con los datos. Para ello se utilizan los datos de los 6 sensores y el mayor número de lecturas posibles por cada muestra.

La normalización manual de los datos se ha eliminado y ahora es el modelo el encargado de preprocesar la normalización de los datos.

Se ha optado por separar en dos funciones la mezcla y la separación.

Mezcla

La función que se muestra a continuación, mezcla las muestras y sus etiquetas correspondientes de cada categoría, mezclando también las categorías.

```python
import random

def shuffle(all_data, all_labels):

    data_samples = len(all_data)

    shuffled_indices = []

    split_attempts = data_samples // categories
    random_indices = [i for i in range(split_attempts)]

    shuffled_categories = [i for i in range(categories)]
    random.shuffle(shuffled_categories)

    for category in shuffled_categories:
        random.shuffle(random_indices)
        for i in random_indices:
            shuffled_indices.append(split_attempts * category + i)

    shuffled_data = all_data[shuffled_indices,:,:]
    shuffled_labels = all_labels[shuffled_indices,:]

    return shuffled_data, shuffled_labels
```

Separación

Separa las muestras y sus etiquetas correspondientes en dos particiones eligiendo la fracción de los datos que se desea para la partición de validación.

```
def split(all_data, all_labels, validation=0.2):

    data_samples = len(data)
    split_attempts = data_samples // categories
    validation_attempts = split_attempts * int((1. - validation) * 100) // 100

    train_indices = []
    validation_indices = []

    indices = [i for i in range(split_attempts)]

    for category in range(categories):
        for i in range(validation_attempts):
            train_indices.append(split_attempts * category + i)
        for i in range(validation_attempts, split_attempts):
            validation_indices.append(split_attempts * category + i)

    x_train = all_data[train_indices,:,:]
    y_train = all_labels[train_indices,:]

    x_validation = all_data[validation_indices,:,:]
    y_validation = all_labels[validation_indices,:]

    return x_train, y_train, x_validation, y_validation
```

6.3.2. Modelo Final

Se ha utilizado un modelo secuencial aunque esté implementado mediante la API funcional de keras, que permite hacer topologías de red más complejas. Para utilizar la API funcional, lo único que hay que saber es que cada capa recibe y devuelve un tensor el cual debe de ser de las mismas dimensiones que la entrada a la capa siguiente.

La topología del modelo final:

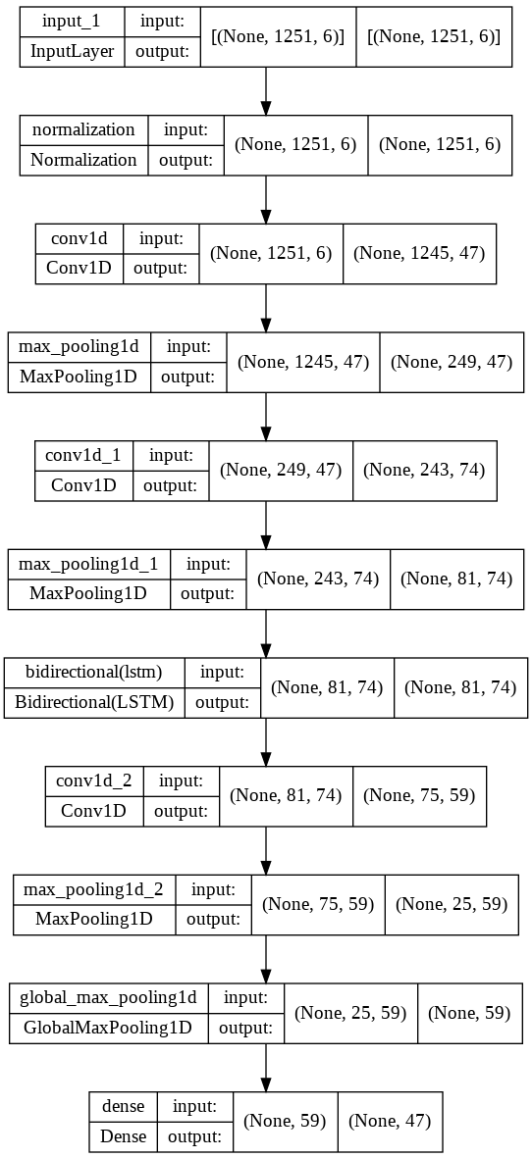


Figura 6.10: Topología de la red neuronal del modelo A

Crear el Modelo

El modelo final, implementado con la API funcional de *Keras*, está compuesto por:

- Una capa de normalización, para tener los datos a la misma escala.
- La base convolucional y recurrente, que se encarga de sacar características de los datos.
- El clasificador, que dependiendo de las características conseguidas en la base convolucional y recurrente asigna una etiqueta de una clase a la muestra de entrada.

```
def create_model(x_train, summary=True):

  # Preprocessing Normalization
  normalizeLayer = Normalization()
  normalizeLayer.adapt(x_train)

  input_tensor = Input(shape=x_train.shape[1:])
  common_input = normalizeLayer(input_tensor)

  # Recurrent and Covolutional Base
  tensor = Conv1D(47, 7, activation=relu)(common_input)
  tensor = MaxPooling1D(5)(tensor)
  tensor = Conv1D(74, 7, activation=relu)(tensor)
  tensor = MaxPooling1D(3)(tensor)
  tensor = Bidirectional(LSTM(37, return_sequences=True))(tensor)
  tensor = Conv1D(59, 7, activation=relu)(tensor)
  tensor = MaxPooling1D(3)(tensor)
  tensor = GlobalMaxPooling1D()(tensor)

  # Classifier
  output_tensor = Dense(47, activation=softmax)(tensor)

  model = Model(input_tensor, output_tensor)

  if summary: model.summary()

  return model
```

Compilación y Entrenamiento del Modelo

La siguiente función compila el modelo y lo entrena, el modelo utilizará de función de pérdida la entropía cruzada que actualizará los pesos o parámetros del modelo utilizando el algoritmo de *backpropagation* y el optimizador *Adam* con la tasa de aprendizaje que ofrece por defecto, que es de 0,001.

Al final el modelo se comporta mejor con 20 épocas o *epochs* y con un tamaño de lote o *batch size* de 32.

Esta función se utiliza tanto para hacer la evaluación *hold-out* como la validación cruzada *k-fold*. Cuando se ejecuta la validación cruzada, ni se guarda automáticamente el modelo mediante el *callback* ni se muestra el progreso de entrenamiento.

```
def compile_and_fit(model, x_train, y_train, x_validation, y_validation,
↪  save_model=False):
    model.compile(loss=categorical_crossentropy,
              optimizer=Adam(),
              metrics=[categorical_accuracy])

    return model.fit(x_train,
                     y_train,
                     epochs = 20,
                     batch_size = 32,
                     callbacks = [threshold, checkpoint, scheduler] if save_model else
                     ↪  [threshold, scheduler],
                     validation_data = (x_validation, y_validation),
                     verbose = 1 if save_model else 0)

```

6.3.3. Evaluación del Modelo

El modelo final fue evaluado de tres formas:

- Observando la matriz de confusión utilizada para mostrar los resultados de la validación *hold-out* [64].

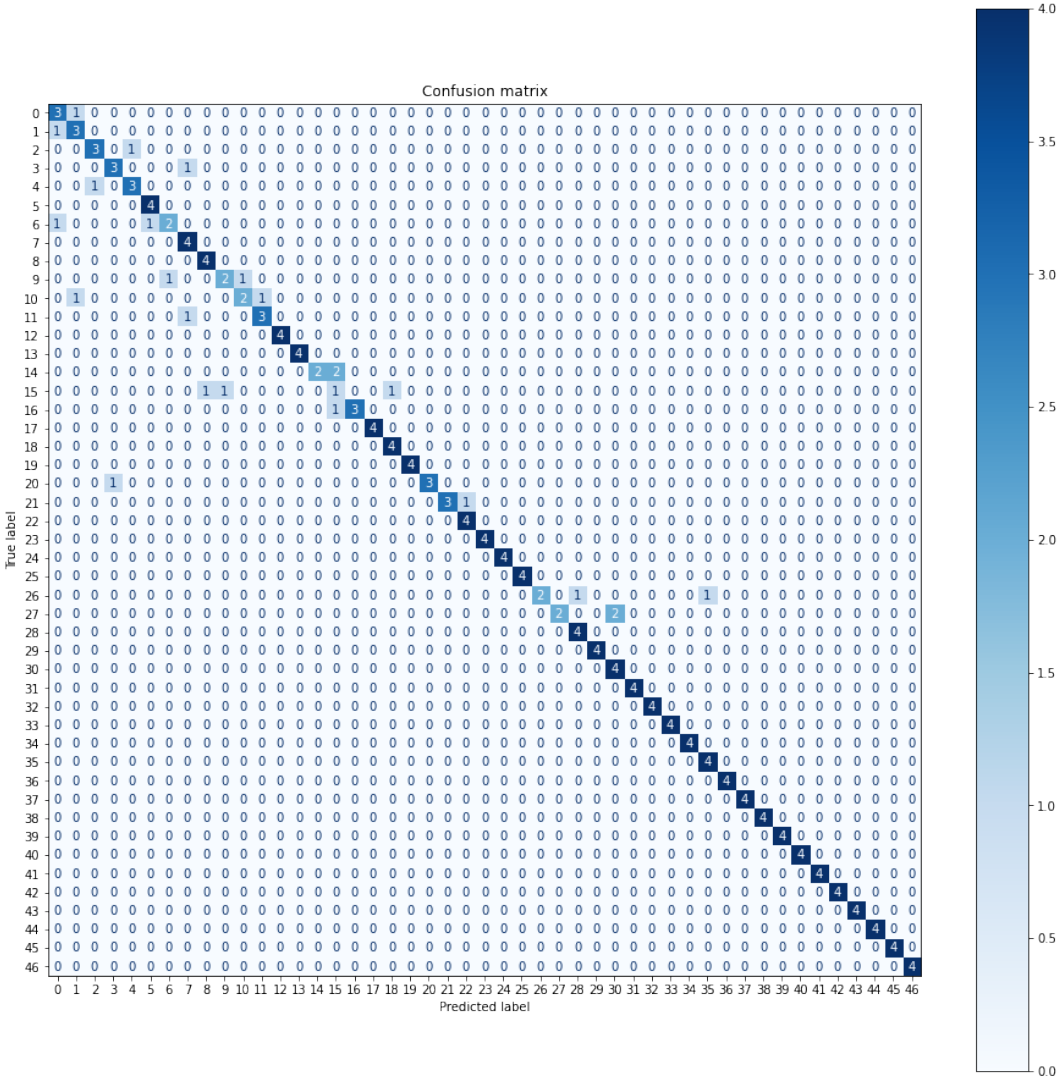


Figura 6.11: Matriz de Confusión - Modelo Final

- Realizando la validación *hold-out*, se obtuvo la precisión y pérdida del modelo frente a los datos de validación y evaluación. Recordar que se realiza una única medida para el modelo al utilizar la validación *hold-out*.
  - Validación
    - Pérdida = 0,444
    - Precisión = 0,881
  - Evaluación
    - Pérdida = 0,473
    - Precisión = 0,872
- Realizando la validación *k-fold* iterativa con mezcla, Donde el número de iteraciones es 2 y k es 8. Hay que tener en cuenta que se empleaban solo las métricas de los datos de la evaluación. Ahora, además muestro las métricas de los datos de la validación [65].
  - Precisión media de la validación = 0,88
  - Precisión media de la evaluación = 0,879

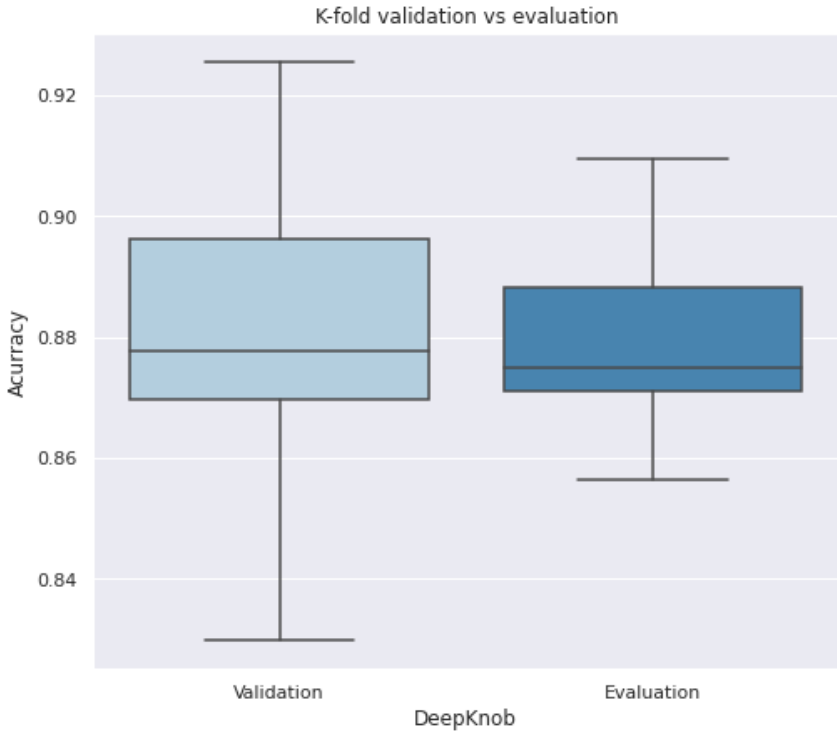


Figura 6.12: Diagrama de caja y bigotes - Modelo final

Capítulo 7

Conclusiones

Dando el proyecto por finalizado, podemos concluir que se han cumplido todos los objetivos del proyecto acorde a los requisitos planteados. A continuación concluiré el problema planteado y la planificación del proyecto, además de dar mi opinión personal sobre este proyecto.

7.1. Problema Planteado

Podemos concluir que es posible identificar a una persona entre 47 diferentes por las secuencias de datos obtenidos por diferentes sensores con una precisión del aproximadamente del 88 %.

Aparte, se ha logrado comparar entre modelos, llegando a elegir el adecuado gracias a la precisión obtenida de realizar un número extenso de medidas para cada modelo. Concluyendo, la validación *k-fold* ayuda a la hora de obtener una medida fiable de la precisión ya que en cada ejecución la precisión puede variar.

Otra conclusión obtenida se debe a que las técnicas de *deep learning* requieren de muchos datos y por lo cual si quitamos datos, ya sean lecturas de la secuencia de sensores como todos los datos obtenidos por un sensor en particular, la precisión bajará.

La ultima conclusión, indica que las redes convolucionales no solo sirven para procesar imágenes y vídeos, sino que también sirven para extraer patrones de secuencias de datos de sensores.

7.2. Planificación

De la planificación podemos concluir que añadir aproximadamente un 15 % del presupuesto al mismo presupuesto puede ser de utilidad en próximos proyectos basados en *deep learning* en los que se empiece sin conocimientos previos.

Para próximos proyectos basados en *deep learning*, gracias a la creación de una guía para resolver problemas de aprendizaje profundo y a la experiencia ganada en este proyecto, los tiempos se acortarán.

7.3. Experiencia personal

Gracias a este proyecto he aprendido nuevas tecnologías, en concreto a utilizar la API *keras* mediante el uso de cuadernos júpiter en la plataforma *google colab*. A parte de entender como utilizar las redes neuronales, también he aprendido como funcionan.

Aunque al comienzo hubo mucha incertidumbre, se logró continuar con el proyecto y finalizarlo. En cuanto a la memoria, opino que es la parte más tediosa y resolver el problema, la parte más entretenida.

7.4. Líneas de trabajo futuras

Se puede investigar si se puede hacer el modelo escalable, que funcione tanto para identificar a usuarios a partir de un grupo de más o menos de 47 individuos. En caso negativo, probar si teniendo menos de 47 individuos identifica correctamente a cualquiera. También se puede probar si hay n individuos tal que $0 < n < 47$ y tratar clasificar a un usuario que no pertenece a ninguna de los n individuos y observar si se clasifica en alguno de las otras $47 - n$ clases.

Además se puede intentar mejorar la precisión del modelo y ver como se comporta con más muestras por usuario.

La última linea de trabajo sería probar con datos nuevos, para ver que tal se comporta el modelo. Si se comporta correctamente, implementar el modelo para ver si es posible su ejecución en tiempo real. Finalmente, si el tiempo y la precisión en tiempo real son viables, implementar un sistema de identificación, por ejemplo en un aula alumnos se podría saber quienes han asistido a clase y comprobando si identifica correctamente a la mayoría de alumnos.

Bibliografía

[1] IBM, "Funcionamiento de svm." https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=models-how-svm-works, 2021-08-17.

[2] P. Baheti, "12 types of neural network activation functions: How to choose?." https://www.v7labs.com/blog/neural-networks-activation-functions, May 17, 2022.

[3] J. Torres, "Introducción práctica con keras." https://torres.ai/deep-learning-inteligencia-artificial-keras/.

[4] F. Chollet, "Deep learning with python," 2018.

[5] C. Yanagisawa, "Conv2d and convtransposed2d." https://indico.cern.ch/event/996880/contributions/4188468/attachments/2193001/3706891/ChiakiYanagisawa_20210219_Conv2d_and_ConvTransposed2d.pdf, 2/19/2021.

[6] I. C. Education, "Gradient descent." https://www.ibm.com/cloud/learn/gradient-descent, 27 Octubre, 2020.

[7] P. Marcelino, "Transfer learning from pre-trained models." https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751, Oct 23, 2018.

[8] F. Berzal, "Entrenamiento de redes neuronales." https://elvex.ugr.es/decsai/deep-learning/slides/NN4%20Training.pdf.

[9] A. G. Muñiz, "Aplicaciones de técnicas de inteligencia artificial basadas en aprendizaje profundo (deep learning) al análisis y mejora de la eficiencia de procesos industriales." https://www.researchgate.net/figure/Figura-27-Ejemplo-de-dropout-A-la-izquierda-se-observa-la-red-original-y-a-la-derecha_fig3_334328255, 2018.

[10] R. Gençay and M. Qi, "Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging." https://www.researchgate.net/publication/3302948_Pricing_and_hedging_derivative_securities_with_neural_networks_Bayesian_regularization_early_stopping_and_bagging/citation/download, 08/2001.

[11] Wikipedia, "Microsoft project." https://es.wikipedia.org/wiki/Microsoft_Project, 13 dic 2021.

[12] Wikipedia, "Microsoft teams." https://es.wikipedia.org/wiki/Microsoft_Teams, 21 nov 2021.

[13] Wikipedia, "Overleaf." https://en.wikipedia.org/wiki/Overleaf, 2 may 2022.

[14] Wikipedia, "Github." https://es.wikipedia.org/wiki/GitHub, 23 may 2022.

[15] Wikipedia, "Google drive." https://es.wikipedia.org/wiki/Google_Drive), 31 may 2022.

[16] Wikipedia, "Python." https://es.wikipedia.org/wiki/Python, 8 jun 2022.

[17] Wikipedia, "Tensorflow." https://es.wikipedia.org/wiki/TensorFlow, 21 oct 2021.

[18] Wikipedia, "Keras." https://es.wikipedia.org/wiki/Keras, 29 ene 2022.

[19] Wikipedia, "Gnu/linux." https://es.wikipedia.org/wiki/GNU/Linux, 4 jun 2022.

[20] Wikipedia, "Anaconda." https://en.wikipedia.org/wiki/Anaconda_(Python_distribution), 17 ene 2022.

[21] Wikipedia, "Proyecto jupyter." https://es.wikipedia.org/wiki/Proyecto_Jupyter), 20 abr 2021.

[22] "Google colaboratory." https://colab.research.google.com.

[23] J. Vegas, C. Llamas, M. A. González, and C. Hernández, "Identifying users from the interaction with a door handle." https://www.journals.elsevier.com/pervasive-and-mobile-computing, 2020.

[24] S. Empiricus, "How to know if model is overfitting or underfitting?." https://stats.stackexchange.com/questions/355774/how-to-know-if-model-is-overfitting-or-underfitting, 2018.

[25] F. Krüger, "Activity, context, and plan recognition with computational causal behaviour models." https://www.researchgate.net/figure/Confusion-matrix-for-multi-class-classification-The-confusion-matrix-of-a_fig7_314116591, Dic 2016.

[26] J. B. Conde, "Análisis y experimentación práctica de frameworks deep learning aplicados a la astronomía." https://uvadoc.uva.es/handle/10324/50088, 2021.

[27] Óscar Fernando Ibáñez Garrido, "Deep learning para análisis de forma en fabricación automática." https://uvadoc.uva.es/handle/10324/50401, 2021.

[28] I. Ivanov Manov, "Caracterización de piezas mediante técnicas de deep learning." https://uvadoc.uva.es/handle/10324/50402, 2021.

[29] A. Trigueros Vega, "Análisis de entrenamiento en deep learning para gestión de calidad en fabricación automática." https://uvadoc.uva.es/handle/10324/50439, 2021.

[30] Wikipedia, "Machine learning." https://en.wikipedia.org/wiki/Machine_learning, 2022.

[31] Wikipedia, "Regression vs. classification in machine learning." https://www.javatpoint.com/regression-vs-classification-in-machine-learning, 2022.

[32] Wikipedia, "Clasificador bayesiano ingenuo." https://es.wikipedia.org/wiki/Clasificador_bayesiano_ingenuo, 22 feb 2022.

[33] A. Pant, "Introduction to logistic regression." https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148, Ene 22, 2019.

[34] Wikipedia, "Árbol de decisión." https://es.wikipedia.org/wiki/%C3%81rbol_de_decisi%C3%B3n, 3 feb 2022.

[35] C. Education, "Random forest." https://www.ibm.com/cloud/learn/random-forest, 7 Diciembre 2020.

[36] Wikipedia, "k vecinos más próximos." https://es.wikipedia.org/wiki/K_vecinos_mas_proximos, 16 dic 2021.

[37] I. C. Education, "Qué son regresión y clasificación en machine learning." https://www.ibm.com/cloud/learn/unsupervised-learning, 21 Septiembre 2020.

[38] Wikipedia, "Sistema de recomendación." https://es.wikipedia.org/wiki/Sistema_de_recomendacion, 2022.

[39] M. T. Jones, "Models for machine learning." https://developer.ibm.com/articles/cc-models-machine-learning/#reinforcement-learning, 4 de diciembre de 2017.

[40] TensorFlow, "Introducción a los tensores." https://www.tensorflow.org/guide/tensor, 2022-01-19.

[41] D. Kumar, "Introduction to data preprocessing in machine learning." https://towardsdatascience.com/introduction-to-data-preprocessing-in-machine-learning-a9fa83a5dc9d, Dic 25, 2018.

[42] Wikipedia, "Deep learning." https://en.wikipedia.org/wiki/Deep_learning, 2022.

[43] B. Marr, "How do you know when and where to apply deep learning?." https://bernardmarr.com/how-do-you-know-when-and-where-to-apply-deep-learning/, 2021.

[44] Wikipedia, "Convolutional neural network." https://en.wikipedia.org/wiki/Convolutional_neural_network, 2022.

[45] F. Ramírez, "Las matemáticas del machine learning: Funciones de activación." https://empresas.blogthinkbig.com/las-matematicas-del-machine-learning-funciones-de-activacion/, 4 junio, 2020.

[46] "Loss functions." https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html, 2017.

[47] N. S. Chauhan, "Optimization algorithms in neural networks." https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html, Dic 18, 2020.

[48] M. Nielsen, "How the backpropagation algorithm works." http://neuralnetworksanddeeplearning.com/chap2.html, Dic 2019.

[49] V. Rodríguez, "Conceptos básicos sobre redes neuronales." https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales, Oct 30, 2018.

[50] Wikipedia, "Learning rate." https://en.wikipedia.org/wiki/Learning_rate, 30 Enero 2022.

[51] S. SHARMA, "Epoch vs batch size vs iterations." https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9, Sep 23, 2017.

[52] A. Oppermann, "Regularization in deep learning — l1, l2, and dropout." https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036, Feb 19, 2020.

[53] J. M. Heras, "Regularización lasso l1, ridge l2 y elasticnet." https://www.iartificial.net/regularizacion-lasso-l1-ridge-l2-y-elasticnet/, 19/09/2020.

[54] M. A. Nadeem and S. U.-J. Lee, "Machine learning evaluation of the requirement engineering process models for cloud computing and security issues." https://www.mdpi.com/2076-3417/10/17/5851/pdf, 2020.

[55] T. Lavanya, N. & Malarvizhi, "Risk analysis and management: a vital key to effective project management." https://www.pmi.org/learning/library/risk-analysis-project-management-7070, 2008.

[56] glassdoor.es, "Sueldos para ingeniero de software junior." https://www.glassdoor.es/Sueldos/ingeniero-de-software-junior-sueldo-SRCH_KO0,28.htm.

[57] "Github." https://github.com.

[58] "Google drive." https://www.google.com/intl/es_es/drive/.

[59] "Keras." https://keras.io.

[60] "Anaconda." https://www.anaconda.com.

[61] "Jupyter." https://jupyter.org.

[62] TensorFlow, "Escribir callbacks de keras personalizados." https://www.tensorflow.org/guide/keras/custom_callback?hl=es-419, 2022-01-24.

[63] Wikipedia, "Confusion matrix." https://en.wikipedia.org/wiki/Confusion_matrix, 19 may 2022.

[64] scikit learn, "sklearn.metrics.confusionmatrixdisplay." https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDisplay.html#sklearn.metrics.ConfusionMatrixDisplay.from_predictions.

[65] Zach, "Seaborn: How to create a boxplot of multiple columns." https://www.statology.org/seaborn-boxplot-multiple-columns/, 31 dic 2021.

Apéndice A

Manuales

Aquí encontrarás las guías de instalación, uso y mantenimiento sobre el proyecto.

A.1. Manual de Instalación y Uso

Para poder que otras personas puedan probar el código, utilizaré *Google Colaboratory* por varias razones:

- Ya están todas las dependencias instaladas.
- Es más intuitivo que usar un cuaderno júpiter en local.
- La persona que lo quiera probar, no tiene porque descargarse nada en local.

A continuación explicaré los simples pasos que hay que seguir para poder ejecutar el cuaderno júpiter en *Google Colaboratory*:

1. Es necesario tener instalado *Google Chrome* y haber iniciado sesión con una cuenta de *Google*.
2. Ir al repositorio de github del proyecto, localizado en el anexo B.
3. Pulsar en el botón *'open in colab'* en el *'readme.md'* o en el archivo *'DeepKnob.ipynb'*.
4. Crear un *zip* llamado *'deepknob-data.zip'* formado por un directorio llamado *'data'* que contiene el *dataset* con el nombre de *'final_dataset.csv'*. Subir el *'deepknob-data.zip'* al almacenamiento de sesión de *Colab*. Figura A.1.

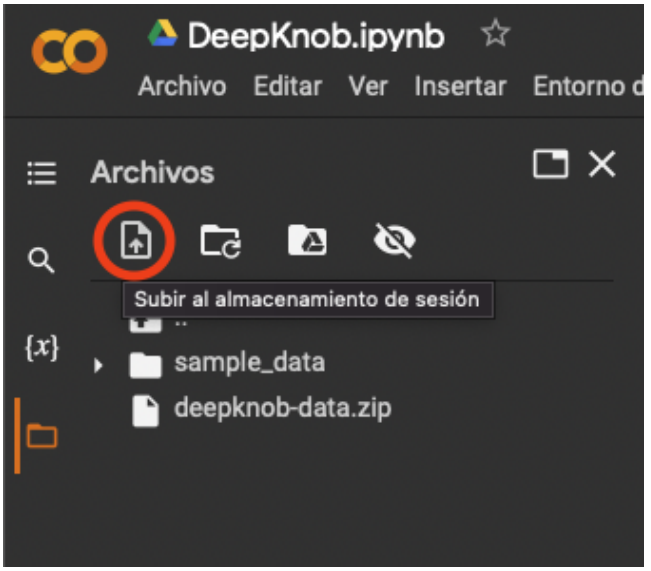


Figura A.1: Subir el *dataset*

5. Ejecutar todas las celdas del cuaderno *DeepKnob.ipynb*. Figura A.2.

Figura A.2: Ejecutar todo el cuaderno

6. Esperar a los resultados.

A.2. Manual de Mantenimiento

La versiones con las que funciona el código actualmente son las de por defecto de *google colab* a día 13/06/22 que consisten de las siguientes versiones:

- keras - 2.8.0
- python - 3.7.13
- numpy - 1.21.6
- matplotlib - 3.2.2
- seaborn - 0.11.2
- pandas - 1.3.5

A.2.1. Preprocesado de datos

Si se cambia de *dataset* se puede hacer fácilmente almacenando el nuevo archivo *csv* en la carpeta *data*. También se pueden cambiar los nombres de los *paths* o caminos al *dataset* en *Dataset* → *Opening the dataset* del cuaderno júpiter.

Figura A.3: Abrir el *dataset*

Siempre que se añadan nuevas muestras al *dataset*, deberán añadirse muestras por clase teniendo el mismo número de muestras por cada clase. Cada muestra debe de estar compuesta por 1495 lecturas de los seis sensores. Preferible que haya 244 lecturas de sensores antes del contacto con la manilla de la puerta.

Una vez mezcladas las muestras, se puede modificar la fracción utilizada para dividir las muestras en dos particiones mediante el método *split* creado.

A.2.2. Modelo

Es posible cambiar el número de usuarios en los que se puede identificar una muestra si se modifican los hiperparámetros de la compilación y entrenamiento del modelo, así como los propios hiperparámetros del modelo.

Modificación de los hiperparámetros

Si se modifican los hiperparámentros del modelo probablemente se tendrá que modificar los hiperparámetros de la compilación y del entrenamiento del mismo modelo. Se podrá cambiar el modelo en *Machine learning approach* → *Model creation* en el cuaderno júpiter. Se pueden modificar otros hiperparámetros como el optimizador, la tasa de aprendizaje, las épocas, el tamaño de lote o los *callbacks* en *Machine learning approach* → *Compile and fit the model*.

Modificación de los *Callbacks*

Para poder modificar cada *callback* creado en *Machine learning approach* → *Model callbacks* se necesita saber la utilidad de cada *callback* 6.2.4. La única variable que cambiaría que afecta a los *callbacks* es *model_path*, también localizada en *Machine learning approach* → *Model callbacks*. La variable *model_path* es la ruta donde se va a guardar el modelo.

A.2.3. Evaluación

La única parte modificable es cuando se ejecuta la función *iterative_k_fold* ya que se puede elegir k y el número de iteraciones ubicado en *Evaluation on test data* → *Iterative k-fold cross validation* → *Iterative k-fold*.

Apéndice B

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código utilizado para la producción `https://github.com/AlfredoFernandezGuaza/deepknob`.
- Documentación API Keras: `https://keras.io/api/`