

Universidades de Burgos, León y  
Valladolid

Máster universitario

# Inteligencia de Negocio y Big Data en Entornos Seguros



**Trabajo Fin de Máster**

**Aplicación de Machine Learning a  
imágenes 1D y 2D de control de  
calidad en recubrimiento de Zinc**

Presentado por Ismael Franco Hernando  
en Universidad de Burgos — 14 de julio  
de 2023

Tutores: Carlos Enrique Vivaracho Pascual  
Joaquín B. Ordieres Meré







## Resumen

En el proceso de galvanizado es muy importante mantener los niveles de la capa de zinc entre unos mínimos, para garantizar que el acero aguante los diferentes escenarios para los que esté dedicado, y unos máximos, para evitar que la empresa pierda dinero al gastar demás de zinc y evitar que el acero se endurezca demasiado evitando que se pueda trabajar con él.

Una empresa dedicada a este proceso, nos ha solicitado que creemos un modelo que, a través de datos medidos por sensores en la cadena de producción, y aplicado sobre bobinas que tiene algún defecto, se realice una predicción sobre si la bobina podrá ser aprovechable o no.

Para ello, se llevarán a cabo diversos experimentos sobre datos 1D y 2D, probando diversos parámetros para obtener el mejor resultado posible.

Además, se llevará a cabo una aplicación que simulará el proceso que un operario de la empresa debería seguir para cargar los datos de las bobinas y esperar las predicciones sobre los mismos.

## Descriptores

Galvanizado, bobinas galvanizadas, *deep learning*, redes neuronales convolucionales, investigación, predicción bobinas, validación cruzada.

## **Abstract**

In the galvanizing process, it is crucial to maintain the levels of the zinc layer within certain minimum and maximum ranges. This is done to ensure that the steel can withstand different scenarios it is intended for, while avoiding excessive zinc usage that could result in financial loss for the company and preventing the steel from becoming too hard to work with.

A company dedicated to this process has requested us to create a model that, using data measured by sensors in the production line and applied to coils with some defects, can predict whether the coil can be salvaged or not.

To achieve this, various experiments will be conducted on 1D and 2D data, testing different parameters to obtain the best possible outcome.

Furthermore, an application will be developed that simulates the process an operator from the company should follow to input the coil data and receive predictions for the same.

## **Keywords**

Galvanized, galvanized coils, deep learning, convolutional neural networks, research, coil prediction, cross-validation.

---

# Índice general

---

<b>Índice general</b>	<b>iii</b>
<b>Índice de figuras</b>	<b>vi</b>
<b>Índice de tablas</b>	<b>viii</b>
<b>1. Introducción</b>	<b>3</b>
1.1. Estructura . . . . .	5
<b>Memoria</b>	<b>3</b>
<b>2. Objetivos del proyecto</b>	<b>7</b>
2.1. Objetivos Generales . . . . .	7
2.2. Objetivos Técnicos . . . . .	7
2.3. Objetivos Personales . . . . .	8
<b>3. Trabajos relacionados</b>	<b>9</b>
3.1. Research on Zinc Layer Thickness Prediction Based on LSTM Neural Network . . . . .	9
3.2. Coating Thickness Modeling and Prediction for Hot-dip Gal- vanized Steel Strip Based on GA-BP Neural Network . . . . .	10
<b>4. Conceptos teóricos</b>	<b>11</b>
4.1. Machine Learning y Deep Learning . . . . .	11
4.2. Redes Neuronales Artificiales . . . . .	12
4.3. Conjunto de Datos Desbalanceado . . . . .	15

<b>5. Técnicas y herramientas</b>	<b>17</b>
5.1. Metodología . . . . .	17
5.2. Herramientas . . . . .	17
5.3. Bibliotecas de Python . . . . .	18
5.4. Documentación . . . . .	20
<b>6. Aspectos relevantes del desarrollo del proyecto</b>	<b>21</b>
6.1. Entendimiento del negocio . . . . .	21
6.2. Entendimiento de los datos . . . . .	22
6.3. Preparación de los datos . . . . .	27
6.4. Modelado . . . . .	29
6.5. Evaluación . . . . .	50
6.6. Despliegue . . . . .	51
<b>7. Conclusiones y Líneas de trabajo futuras</b>	<b>53</b>
7.1. Conclusiones . . . . .	53
7.2. Líneas de trabajo futuras . . . . .	54
<b>Apéndices</b>	<b>56</b>
<b>Apéndice A Plan de Proyecto Software</b>	<b>59</b>
A.1. Introducción . . . . .	59
A.2. Planificación temporal . . . . .	59
A.3. Estudio de viabilidad . . . . .	59
<b>Apéndice B Especificación de Requisitos</b>	<b>63</b>
B.1. Introducción . . . . .	63
B.2. Objetivos generales . . . . .	63
B.3. Catalogo de requisitos . . . . .	63
B.4. Especificación de requisitos . . . . .	64
<b>Apéndice C Especificación de diseño</b>	<b>69</b>
C.1. Introducción . . . . .	69
C.2. Diseño de datos . . . . .	69
C.3. Diseño procedimental . . . . .	71
C.4. Diseño arquitectónico . . . . .	71
<b>Apéndice D Documentación técnica de programación</b>	<b>73</b>
D.1. Introducción . . . . .	73
D.2. Estructura de directorios . . . . .	73
D.3. Manual del programador . . . . .	75



<i>Índice general</i>	v
D.4. Compilación, instalación y ejecución del proyecto . . . . .	76
D.5. Pruebas del sistema . . . . .	77
<b>Apéndice E Documentación de usuario</b>	<b>79</b>
E.1. Introducción . . . . .	79
E.2. Requisitos de usuarios . . . . .	79
E.3. Instalación . . . . .	79
E.4. Manual del usuario . . . . .	80
<b>Bibliografía</b>	<b>83</b>

---

## Índice de figuras

---

4.1. Esquema Algoritmos Machine Learning (Realización Propia). . .	12
4.2. Ejemplo Red Neuronal Artificial (Realización Propia) . . . . .	13
4.3. Operación de Convolución [3] . . . . .	15
4.4. Operación de <i>Pooling</i> [4] . . . . .	15
6.5. Ejemplo estructura datos 1D . . . . .	23
6.6. Ejemplo estructura datos 2D . . . . .	23
6.7. Ejemplo tabla GR_LPValues . . . . .	25
6.8. Ejemplo tabla GR_QPValues . . . . .	25
6.9. Ejemplo tabla V_Coilss . . . . .	26
6.10. Ejemplo de Visualización para una Bobina . . . . .	26
6.11. Ejemplo de características obtenidas para datos 1D . . . . .	28
6.12. Ejemplo de características obtenidas para datos 2D . . . . .	28
6.13. Ejemplo de Construcción de Modelo . . . . .	30
6.14. Matriz de Confusión Experimento 1 . . . . .	35
6.15. Matriz de Confusión Experimento 2 . . . . .	36
6.16. Matriz de Confusión Experimento 3 . . . . .	36
6.17. Matriz de Confusión Experimento 4 . . . . .	37
6.18. Matriz de Confusión Experimento 5 . . . . .	37
6.19. Matriz de Confusión Experimento 6 . . . . .	38
6.20. Matriz de Confusión Experimento 7 . . . . .	38
6.21. Matriz de Confusión Experimento 8 . . . . .	39
6.22. Matriz de Confusión Experimento 9 . . . . .	39
6.23. Matriz de Confusión Experimento 10 . . . . .	40
6.24. Matriz de Confusión Experimento 11 . . . . .	40
6.25. Matriz de Confusión Experimento 12 . . . . .	41
6.26. Matriz de Confusión Experimento 13 . . . . .	42
6.27. Matriz de Confusión Experimento 14 . . . . .	42

6.28. Matriz de Confusión Experimento 15 . . . . .	43
6.29. Matriz de Confusión Experimento 16 . . . . .	43
6.30. Matriz de Confusión Experimento 17 . . . . .	44
6.31. Matriz de Confusión Experimento 18 . . . . .	44
6.32. Matriz de Confusión Experimento 19 . . . . .	45
6.33. Matriz de Confusión Experimento 20 . . . . .	45
6.34. Matriz de Confusión Experimento 21 . . . . .	46
6.35. Matriz de Confusión Experimento 22 . . . . .	46
6.36. Matriz de Confusión Experimento 23 . . . . .	47
6.37. Matriz de Confusión Experimento 24 . . . . .	47
6.38. Matriz de Confusión Experimento 25 . . . . .	48
6.39. Matriz de Confusión Experimento 26 . . . . .	49
6.40. Matriz de Confusión Experimento 27 . . . . .	49
6.41. Matriz de Confusión Experimento 28 . . . . .	50
6.42. Ejemplo de Resultado de la Aplicación . . . . .	51
6.43. Ejemplo de Resultado del CSV Genrado por la Aplicación . . .	52
6.44. Ejemplo de Resultado del histroial.txt Genrado por la Aplicación	52
 B.1. Diagrama de Casos de Uso . . . . .	 65
 C.1. Diagrama de Secuencia . . . . .	 71
 E.1. Ejecución de la Primera Celda . . . . .	 81
E.2. Ejecución de la Tercera Celda . . . . .	81

---

# Índice de tablas

---

A.1. Costes <i>hardware</i> estimados . . . . .	60
A.2. Costes personal estimados . . . . .	60
A.3. Costes totales estimados . . . . .	61
A.4. Licencias Bibliotecas Python y Herramientas . . . . .	61
B.1. Caso de Uso - 1 . . . . .	66
B.2. Caso de Uso - 2 . . . . .	66
B.3. Caso de Uso - 3 . . . . .	67
B.4. Caso de Uso - 4 . . . . .	67

# Memoria



---

# Introducción

---

El galvanizado [5] es una técnica que se emplea para darle al acero una protección frente a la corrosión, producida principalmente por el aire y la humedad, de forma que estas causan una reacción química o electroquímica, produciendo el deterioro y oxidación del acero.

Esta técnica consiste en bañar las piezas de acero en zinc, ya que este proporciona al metal, no solo protección frente a la corrosión, sino que también mejora su resistencia a golpes y a la abrasión. Esto permite que este tipo de acero sea idóneo para usarse en el exterior o en lugares con mucha humedad o especialmente corrosivos.

El proceso de galvanizado puede ser de dos tipos:

- **Galvanizado en caliente:** se introduce la pieza de acero en un recipiente que contiene zinc fundido a aproximadamente 450 °C y que se deja hasta que su recubrimiento alcanza las micras deseadas. Generalmente, si se desea que el recubrimiento no dure demasiado tiempo, la capa de zinc tiene un tamaño de entre 7 y 45 micras, y si, en cambio, se desea que el recubrimiento dure más tiempo, el tamaño de la capa de zinc suele estar entre las 45 y las 200 micras.
- **Galvanizado en frío:** para este tipo de galvanizado se aplica zinc mediante pulverización o con electrodeposición hasta que la capa de zinc tiene un tamaño de entre 5 y 20 micras. Este tipo de galvanizado se utiliza generalmente para piezas más pequeñas o incluso para usos de interior, ya que otorga una mejor estética.

Cabe destacar que esta técnica se usa para la industria de la construcción, y principalmente en aquella dedicada a las piezas metálicas como tuberías o

barandillas. Otras de las principales industrias en la que se emplea este tipo de acero es en automoción y en la producción de electrodomésticos. Como se puede ver, las ramas de uso del acero galvanizado son muy variadas, y todo ello debido a sus principales ventajas:

- **Durabilidad y resistencia:** el bañado en zinc del acero aguanta durante una elevada cantidad de años, además de otorgar a la pieza una protección extra frente a los golpes.
- **Protección a la corrosión:** siendo esta una de las principales ventajas y uso de este tipo de acero, ya que evita que la pieza se corroa, alargando la vida útil, soportando incluso los ambientes con mucha humedad y altamente corrosivos.
- **Mantenimiento:** aunque el proceso de galvanizado puede suponer un coste elevado en la fabricación, la durabilidad de este material sin la necesidad de ningún tipo de mantenimiento hace que sea muy cómodo su utilización.
- **Gran versatilidad:** el galvanizado se puede aplicar tanto a piezas grandes, como por ejemplo bobinas con elevadas longitudes de acero, como a piezas pequeñas, como tuercas y tornillos. Además, independientemente de la forma que tenga, se podrá aplicar este proceso, consiguiendo así que el galvanizado se pueda aplicar a multitud de piezas.
- **Fiabilidad:** en la actualidad existen varias leyes que han regulado el acero galvanizado, de manera que las piezas deben de cumplir unos estándares, garantizando así que la pieza cumpla con unos mínimos.

Una vez visto qué es el galvanizado, cómo funciona y cuáles son sus principales ventajas, es turno de introducir el origen de este proyecto. Una empresa dedicada a la producción de bobinas galvanizadas, cuyo nombre, por confidencialidad no se puede incluir en esta memoria, nos ha solicitado desarrollar un modelo que sea capaz de identificar bobinas aprovechables, dentro de aquellas que están descatalogadas por no cumplir con la cantidad de zinc necesaria para garantizar un correcto galvanizado.

La producción de estas bobinas de acero [6] es difícil, ya que en primer lugar es necesario un tratamiento de limpieza y un tratamiento térmico previo al recubrimiento de zinc. Además, el acero siempre tiene que estar entre unos mínimos de zinc, para garantizar un correcto galvanizado y evitar



que el material empiece a corroerse antes de tiempo, y entre un máximo, puesto que la empresa estaría perdiendo dinero y además el acero obtenido podría ser demasiado duro impidiendo su uso por parte los clientes. Es por ello, que cuentan con varios sensores que miden la cantidad de zinc que tiene el acero y que identifica aquellas bobinas que no cumplen con los criterios de zinc necesarios.

Pese a que haya bobinas que no cumplan con las necesidades de zinc exigidas por el cliente, puede haber ciertas partes aprovechables, por lo que en la actualidad la empresa utiliza un modelo que, aproximadamente, es capaz de identificar correctamente tan solo un 10 % de las bobinas, y el otro 90 % restante depende de un operario que analiza una a una e identifica aquellas que puedan servir. Este proceso es demasiado tedioso y toma mucho tiempo al operario, por lo que se buscará obtener algún modelo que sea capaz de identificar de manera correcta la mayoría de bobinas aprovechables por parte de la empresa, consiguiendo mejorar considerablemente los tiempos de análisis.

## 1.1. Estructura

La documentación de este proyecto está formado por un único documento que esta dividido en dos grandes bloques: Memoria y Apéndices, cuya estructura es la siguiente:

### Memoria

La estructura de la memoria está compuesta por los capítulos siguientes:

1. **Introducción:** pone en contexto al lector sobre el problema al que se enfrenta el proyecto. Además, contiene la estructura de toda la documentación. Actualmente el lector se encuentra en este capítulo.
2. **Objetivos del Proyecto:** contiene los objetivos generales, técnicos y personales marcados en el proyecto.
3. **Trabajos Relacionados:** habla sobre algunos artículos y trabajos que también están relacionados con el galvanizado y la obtención de modelos basados en el *deep learning*.
4. **Conceptos Teóricos:** explicación al lector de los conceptos necesarios para comprender correctamente el proyecto y sus diversas justificaciones.

5. **Técnicas y Herramientas:** conjunto de técnicas, metodologías y herramientas que se han utilizado durante el transcurso del proyecto.
6. **Aspectos Relevantes del Desarrollo del Proyecto:** describe y explica claramente todo el proceso llevado a cabo durante el proyecto, junto con los problemas y soluciones que se han encontrado.
7. **Conclusiones y Líneas de Trabajo Futuras:** tiene la conclusión final del proyecto y como podría seguir evolucionando el proyecto con nuevas mejoras de cara al futuro.

## Anexo

El apéndice está formado por un total de cinco puntos:

1. **Plan de Proyecto:** contiene la planificación seguida junto con sus viabilidades económicas y legales.
2. **Especificación de Requisitos:** describe los diversos objetivos generales, requisitos y sus correspondientes caso de uso.
3. **Especificación de Diseño:** describe los datos, junto con su diseño procedimental y arquitectónico.
4. **Documentación Técnica de Programación:** describe la estructura de directorios, el manual del programador, la ejecución del proyecto y las diversas pruebas realizadas al sistema.
5. **Documentación de Usuario:** contiene los requisitos de usuario, la instalación y el manual de usuario necesarios para poder utilizar correctamente la aplicación del proyecto.

---

# Objetivos del proyecto

---

Este apartado contiene los diferentes objetivos que se han marcado durante el desarrollo del proyecto. En total se distinguen tres tipos: generales, técnicos y personales.

## 2.1. Objetivos Generales

Los objetivos generales marcados en el proyecto son:

- Crear atributos que describan la calidad del producto y puedan ser utilizados posteriormente como atributos para modelar.
- Evaluar diferentes estrategias de modelado con técnicas de *machine learning* y *deep learning* para realizar recomendaciones.
- Realizar una aplicación utilizable por un operario para validar diferentes bobinas.

## 2.2. Objetivos Técnicos

Los objetivos técnicos marcados en el proyecto son:

- Desarrollar y entrenar un modelo de *TensorFlow* que sea capaz de identificar el mayor número de bobinas válidas y no válidas para su uso.
- Configurar y optimizar los parámetros del modelo con el fin de conseguir la mayor precisión posible.

- Preprocesar el conjunto de datos proporcionado por la empresa para que el modelo que se genere pueda identificar de la mejor forma posible las propiedades más relevantes.

## 2.3. Objetivos Personales

Los objetivos personales marcados en el proyecto son:

- Aplicar los conocimientos adquiridos a lo largo del máster universitario.
- Profundizar, mejorar y aplicar los conocimientos sobre el lenguaje de programación *Python*.
- Acercarse a la práctica profesional de las enseñanzas del máster.

---

## Trabajos relacionados

---

Analizando trabajos que también usen el *deep learning* en la rama de la industria centrada en el acero galvanizado, existe una gran variedad de artículos centrados en este tema; lo que afirma la importancia de controlar los niveles de zinc en el acero, comentado previamente en la introducción.

Pese a todo, el número de artículos centrados en los mismos objetivos que este proyecto es bastante limitado, ya que la mayoría busca predecir las capas de zinc de láminas de acero con redes neuronales. A continuación se muestran dos artículos, cuyo principal objetivo es predecir las capas de zinc del acero galvanizado, a partir de diferentes medidas tomadas en la línea de producción.

### 3.1. Research on Zinc Layer Thickness Prediction Based on LSTM Neural Network

Autores: Zhao Lu, Yimin Liu y Shi Zhong

Este artículo [9] utiliza una red neuronal LSTM (*Long Short-Term Memory*), que es un tipo de red recurrente (y que en el siguiente punto se explicará más en detalle), que permite detectar propiedades de los datos a lo largo del tiempo gracias a su capacidad de retener memoria.

El propósito de dicho artículo es el de predecir el espesor del zinc de acero galvanizado en caliente, con el fin de ver si cumple con los requisitos mínimos y máximos necesarios. Para ello se emplearán los diferentes datos

medidos por los sensores en la cadena de producción, y, en última instancia, se realizará la predicción.

Finalmente, los resultados obtenidos son muy buenos, ya que su error porcentual absoluto es del 1.824 % y teniendo en cuenta que las capas de zinc se miden en micras, son un valor muy bueno debido a la alta precisión que se necesita tener.

### **3.2. Coating Thickness Modeling and Prediction for Hot-dip Galvanized Steel Strip Based on GA-BP Neural Network**

Autores: Kai Mao, Yong-Li Yang, Zhe Huang y Dan-yang Yang

Este segundo artículo [10] es muy parecido al anterior, solo que utilizan una red neuronal BP (*Back Propagation*), que se caracteriza por ser una red que puede ajustar con el paso del tiempo los pesos de sus enlaces con el fin de reducir al máximo la función de error.

El principal objetivo del artículo es predecir el espesor de zinc sobre chapas de acero galvanizadas en caliente. Para ello, usarán diferentes parámetros medidos durante el proceso de galvanizado, como la velocidad de la línea, la presión de la chapa que corta el acero o la temperatura a la que se encuentra el zinc.

Finalmente, se comenta que los resultados de la red neuronal BP no son demasiado buenos, por lo que se añadió también un algoritmo genético que permitió optimizar la red, y con ello obtener buenos resultados; comentando incluso que es posible su uso por parte de las empresas en la fase de control de calidad.

---

## Conceptos teóricos

---

### 4.1. Machine Learning y Deep Learning

El proyecto se ha basado en el uso del *machine learning*, y más en concreto, en una de sus ramas: el *deep learning*.

#### Machine Learning

El aprendizaje automático, también conocido como *machine learning* en inglés, [2] se centra en conseguir que con unos datos iniciales, y aplicado algún tipo de algoritmo o técnica, un computador sea capaz de aprender.

Para conseguir esto, adopta el proceso de observación y aprendizaje utilizado por los seres vivos, de forma que cuantos más “sucesos” haya, mayor conocimiento podrá obtener, es decir, cuanto mayor sea el número de datos con el que se cuente, los resultados obtenidos también serán mejores. En la Figura 4.1 se puede ver detalladamente como funciona el *machine learning*.

Este tipo de aprendizaje debe utilizarse en problemas donde se tengan multitud de datos y multitud de escenarios y posibilidades, y que a su vez se puedan modelar todas sus soluciones de forma matemática.

Finalmente, los tipos de *machine learning*, según el problema al que se enfrente, son:

- **Aprendizaje supervisado:** cada dato del conjunto de datos va acompañado de la clase a la que pertenece.
- **Aprendizaje no supervisado:** se desconoce la clase a la que pertenece cada dato.

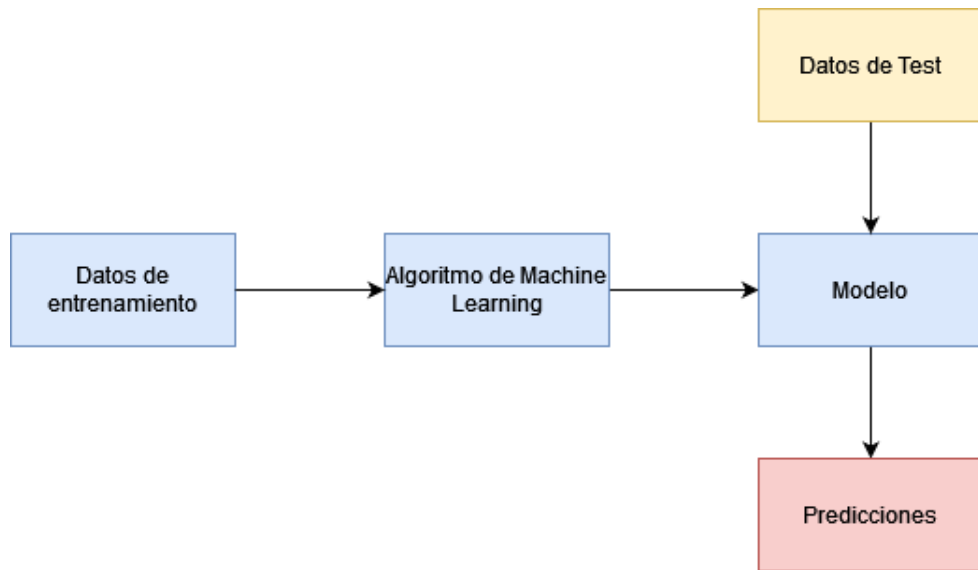


Figura 4.1: Esquema Algoritmos Machine Learning (Realización Propia).

- **Aprendizaje semisupervisado:** en el conjunto de datos, se conocerá la clase de alguno de los datos, mientras que de otros no.
- **Aprendizaje por refuerzo:** el algoritmo aprende en base a sus aciertos y fallos, mediante un sistema de recompensas y penalizaciones.

## Deep Learning

El aprendizaje profundo, también conocido como *deep learning* en inglés, [2] es una de las ramas del *machine learning* que se basa en el sistema nervioso humano, donde las neuronas están conectadas entre sí y cada una de ellas se encarga de una tarea en específico.

Esta estructura permite que ciertas zonas se encarguen de detectar ciertos patrones o características de los datos, lo que permite una gran mejoría con respecto al *machine learning*, ya que esta estructura otorga a la red la posibilidad de aprender de manera automática, además de conseguir mejoras de rendimiento.

## 4.2. Redes Neuronales Artificiales

Las redes neuronales artificiales [8] buscan recrear el sistema nervioso del ser humano, donde existen multitud de neuronas conectadas entre sí, y que intercambian pulsos.



En una red neuronal artificial existen multitud de neuronas artificiales conectadas entre sí por un enlace, que habitualmente suele llevar un peso asociado, consiguiendo así que otra neurona se active según el pulso que le llegue por sus enlaces.

Durante el proceso de entrenamiento, y de manera iterativa, se van modificando estos pesos, buscando los valores óptimos que minimicen alguna función de error o pérdida. Es la forma en la que red “aprende” a partir de los datos que se le prestan.

Según como fluyan los datos en la red, se distinguen principalmente dos tipos:

- **Red neuronal prealimentada:** los datos siempre van hacia delante pasando por diferentes capas hasta llegar a la capa de salida. Durante todo este proceso no se permiten ni bucles ni volver hacia atrás, es decir, la información siempre se mueve hacia la siguiente capa. La Figura 4.2 muestra de una manera más gráfica cómo se mueve la información. Este tipo de red es el más básico y se usa principalmente en problemas de clasificación y regresión.
- **Red neuronal recurrente:** permite que a lo largo de la red haya bucles o que se pueda avanzar hacia una capa anterior, otorgando a las neuronas una memoria interna que permitirá ir adaptándose según lleguen nuevos datos. Es por ello que se utiliza principalmente en problemas con datos secuenciales o *data stream*.

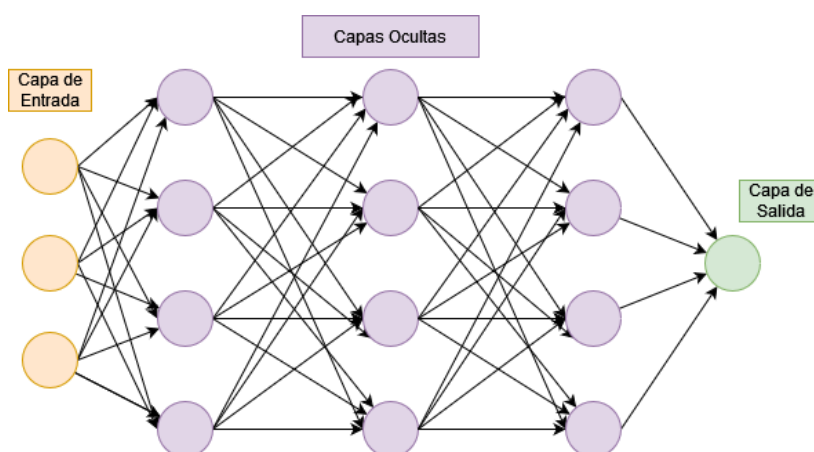


Figura 4.2: Ejemplo Red Neuronal Artificial (Realización Propia)

## Redes Neuronales Convolucionales

Las redes neuronales convolucionales [12], también conocidas como CNN (por sus siglas en inglés *Convolutional Neural Network*), son la variante de las redes neuronales artificiales más extendidas y utilizadas en la actualidad. Esto se debe al gran rendimiento que se obtiene en aquellos casos donde los datos se encuentren en cuadrícula, como un *array* o matriz. Sus principales usos son en la clasificación de imágenes o en la detección de objetos. Es por ello, que, pese a que existan muchas más variantes de redes neuronales, se ha empleado este tipo durante el desarrollo del proyecto.

Este tipo de red se basa en aplicar dos operaciones diferentes: convolución y *pooling*.

### Convolución

La convolución [12] es una operación que busca obtener características de los datos para que puedan ser aprendidos por la red neuronal. Para ello se utiliza una matriz, comúnmente denominada *kernel*, que recorre todas las celdas de los valores de entrada y calcula, junto con las celdas de su alrededor, el producto escalar y dicho valor se almacena en el *kernel*. De esta forma se va completando, y obteniendo una matriz, generalmente, de un tamaño menor que el de los datos de entrada.

En la Figura 4.3 se puede ver como funciona esta operación usando un *kernel* de tamaño 3x3 sobre los datos iniciales.

### Pooling

La operación *pooling* [12] busca reducir la dimensionalidad de la matriz de características obtenida en la operación anterior, de forma que queden aquellas más relevantes y se disminuya la carga computacional de la red neuronal.

Para ello, se divide el *kernel* en una malla de un tamaño en específico, en la que, para cada celda, se devuelve un valor, que habitualmente suele ser el valor máximo o el valor medio; ya que según el problema a resolver puede ser mejor solución el uso del valor máximo, puesto que resalta las características más importantes, o el valor medio, que devuelve el promedio de las características.

En la Figura 4.4 se puede observar un ejemplo de *pooling* aplicando una malla de 2x2 y devolviendo por cada celda el valor máximo.

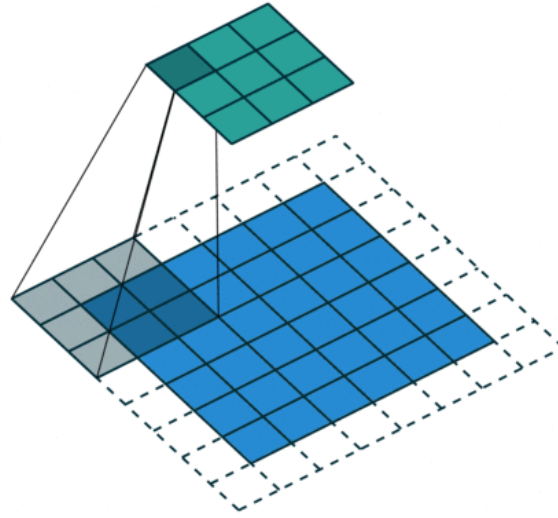
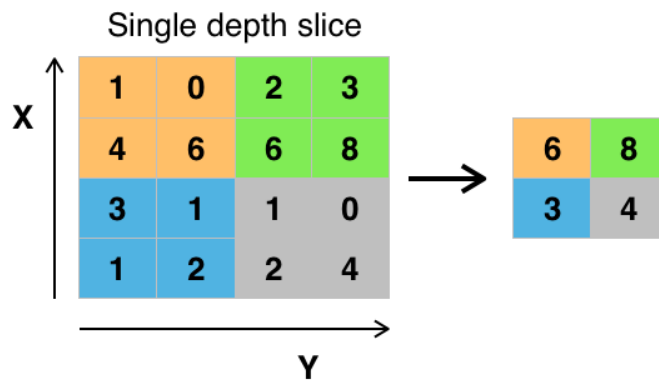


Figura 4.3: Operación de Convolución [3]

Figura 4.4: Operación de *Pooling* [4]

### 4.3. Conjunto de Datos Desbalanceado

En problemas de clasificación, cuando se quiere construir un modelo aplicando *deep learning* es importante contar con una gran cantidad de datos de cada una de las clases. Aunque esto es, en algunos casos, difícil, por ejemplo, en medicina es más común que la gente esté sana a que la gente esté enferma, por lo que en este conjunto de datos sería difícil de tener un número de personas equilibrado de ambas clases o el número de muestras equitativo de ambas clases sería limitado. A esto se le conoce como conjunto de datos desbalanceado o desequilibrado.

Ante estos inconvenientes, suele ser recomendable y eficiente aplicar alguna de las tres estrategias siguientes [11]:

- **Submuestreo:** consiste en reducir el número de datos de la clase mayoritaria para tener ambas clases con el mismo número de registros.
- **Sobremuestreo:** se basa en generar nuevos datos de la clase minoritaria para poder equilibrar los datos de cada clase. Estos datos pueden ser repitiendo los mismos datos o creando nuevos “sintéticos”, ya que según el problema puede ser mejor utilizar una técnica u otra.
- **Sobremuestreo y submuestreo:** se aplica en primer lugar el sobremuestreo para equilibrar ambas clases, y a continuación, aplicar el submuestreo sobre todos los datos para eliminar aquellos que estén repetidos o sean muy similares.

---

# Técnicas y herramientas

---

Este punto muestra las diferentes técnicas y herramientas que se han utilizado en el proyecto. Además, también incluye la metodología y gestión seguida en la construcción del proyecto.

## 5.1. Metodología

Durante el desarrollo del proyecto se ha empleado la metodología CRISP-DM [7], es decir, se han realizado diferentes etapas (entendimiento, preparación, modelado, evaluación y despliegue) con el conjunto de datos original.

Además, en alguna de las etapas se ha empleado la metodología ágil *Scrum*. En dicha metodología [1] se pretende efectuar las diferentes tareas de manera incremental, formando un *sprint*.

Finalmente, indicar que se han llevado a cabo reuniones cada dos semanas para revisar el progreso y tomar las decisiones adecuadas en el correcto desarrollo del proyecto.

## 5.2. Herramientas

### Anaconda

*Anaconda*<sup>1</sup> es una distribución de uso libre y abierta basada en *Python* y cuyo principal uso es en las ramas de ciencias de datos y en aprendizaje automático.

---

<sup>1</sup>Anaconda: <https://www.anaconda.com/>

## Jupyter Notebook

*Jupyter Notebook*<sup>2</sup> es una interfaz web utilizada principalmente para ejecutar sentencias de código desde el navegador a través de los *notebooks* que permite generar.

## GitHub

*GitHub*<sup>3</sup> es un servicio que permite alojar los proyectos de manera remota, e ir subiendo las nuevas versiones que se desarrollen de manera que se pueda llevar un control de las diferentes versiones, con sus cambios y mejoras.

## 5.3. Bibliotecas de Python

Durante todo el desarrollo del proyecto se ha utilizado el lenguaje de programación *Python* en diferentes *notebooks*. Las bibliotecas que se han empleado durante el desarrollo han sido las siguientes.

### Pandas

La librería *Pandas*<sup>4</sup> permite manipular y analizar grandes cantidades de datos. Se asemeja mucho a la herramienta de *Excel* pero en *Python*.

### NumPy

*NumPy*<sup>5</sup> permite utilizar arrays y matrices sobre las cuales realizar las principales operaciones matemáticas. Además, son muy útiles frente a grandes conjuntos de datos, ya que las operaciones están muy optimizadas.

### MySQL

La librería *MySQL*<sup>6</sup> permite conectarse a una base de datos *MySQL* desde *Python*. Además, permite realizar las principales operaciones en una base de datos tradicional: consultas y añadir o eliminar datos.

---

<sup>2</sup>Jupyter Notebook: <https://jupyter.org/>

<sup>3</sup>GitHub: <https://github.com/>

<sup>4</sup>Pandas: <https://pandas.pydata.org/>

<sup>5</sup>NumPy: <https://numpy.org/>

<sup>6</sup>MySQL: <https://www.mysql.com/>

## PyMySQL

La librería *PyMySQL* es muy similar a la anterior, solo que permite llevar a cabo operaciones en bases de datos de forma mucho más amigable por parte del usuario.

## TensorFlow

*TensorFlow*<sup>7</sup> es una librería desarrollada por *Google* y que se centra principalmente en el uso del aprendizaje automático y la inteligencia artificial. Siendo el primer caso, el utilizado en este proyecto, para poder configurar y crear modelos de aprendizaje automático.

Además, incluye *Keras*<sup>8</sup> lo que simplifica y facilita la construcción de redes neuronales.

## Scikit-Learn

La librería *Scikit-learn*<sup>9</sup> se centra en el aprendizaje automático, incluyendo los principales algoritmos de clasificación y regresión. Además, incluye alguna herramienta que permite valorar los resultados obtenidos con el modelo, siendo este punto la principal utilidad en el proyecto.

## Imbalanced-Learn

*Imbalanced-learn*<sup>10</sup> se centra en abordar el problema de clases desbalanceadas, de modo que tiene múltiples herramientas para intentar de solventar este inconveniente.

## Matplotlib

*Matplotlib*<sup>11</sup> permite generar los gráficos más habituales en dos dimensiones, además, estas pueden ser interactivas o dinámicas, facilitando así la comprensión por parte del usuario.

---

<sup>7</sup>TensorFlow:<https://www.tensorflow.org>

<sup>8</sup>Keras:<https://keras.io/>

<sup>9</sup>Scikit-learn:<https://scikit-learn.org/>

<sup>10</sup>Imbalanced-learn:<https://imbalanced-learn.org/>

<sup>11</sup>Matplotlib: <https://matplotlib.org/>

## Jupyter Widgets

*Jupyter Widgets*<sup>12</sup> otorga a los *notebooks* un cierto dinamismo e interactividad al usuario, de forma que los *notebooks* no queden tan estáticos, y que no suponga únicamente ejecutar celdas sin apenas interacción por parte del usuario.

## 5.4. Documentación

### L<sup>A</sup>T<sub>E</sub>X

L<sup>A</sup>T<sub>E</sub>X<sup>13</sup> es un sistema de composición de textos usado principalmente en la escritura de textos con alta calidad tipográfica.

### Overleaf

*Overleaf*<sup>14</sup> es un editor colaborativo de L<sup>A</sup>T<sub>E</sub>X cuyo principal uso es escribir, editar y publicar documentos científicos, todo ello gestionado desde la nube.

### Draw.io

*Draw.io*<sup>15</sup> es una herramienta web que permite, entre otras cosas, crear diagramas, por lo que se ha empleado principalmente para representar pequeños esquemas o diagramas UML.

### Tables Generator

*Tables Generator*<sup>16</sup> es una web que facilita la creación de tablas para L<sup>A</sup>T<sub>E</sub>X, ya que se pueden diseñar a través de una interfaz y posteriormente exportarlo a código.

---

<sup>12</sup>Jupyter Widgets: <https://github.com/jupyter-widgets/ipywidgets>

<sup>13</sup>L<sup>A</sup>T<sub>E</sub>X: <https://www.latex-project.org/>

<sup>14</sup>Overleaf: <https://www.overleaf.com/>

<sup>15</sup>Draw.io: <https://app.diagrams.net/>

<sup>16</sup>Tables Generator: <https://www.tablesgenerator.com/#>



---

# Aspectos relevantes del desarrollo del proyecto

---

Este punto contiene, de manera descriptiva y detallada, el proceso seguido desde el inicio hasta el fin del proyecto.

Cómo ya se ha comentado en el punto anterior de la memoria, la metodología utilizada ha sido CRISP-DM, y es por ello que el orden de este punto se debe a las diferentes fases. A continuación, se describe detalladamente cada una de las 6 fases.

## 6.1. Entendimiento del negocio

En esta primera fase se ha buscado información sobre el proceso de galvanizado, con el fin de ponerse en contexto con el tema del proyecto y entender a que se dedica la empresa que da origen a este trabajo.

A continuación, se puso en contexto el objetivo principal del proyecto, que es, sobre unos datos tomados por sensores en la cadena de producción, crear un modelo capaz de predecir si una bobina podrá ser aprovechable por parte de la empresa o no, en función de si el recubrimiento de zinc cumple unos estándares.

Los sensores toman datos de cada una de las *tejas* de la bobina. Una *teja* es una porción de bobina, de entre unos 10 y 25 metros (según el tipo de sensor), sobre la que se mide la capa de zinc para posteriormente almacenar su valor.

Además, cada bobina tendrá una cantidad de zinc mínimo y máximo estipulado que tendrá que cumplir para que pueda ser utilizada en el escenario para la que se está construyendo.

## 6.2. Entendimiento de los datos

La empresa nos ha proporcionado datos que han sido tomados en la cadena de producción, y que han sido evaluados previamente por un operario, otorgándoles dos posibles clases:

- **OK:** la bobina es válida y puede ser usada por la empresa.
- **NOK:** la bobina no cumple con los requisitos del cliente y/o la empresa y, por tanto, será descartada.

Cabe destacar que una bobina, a pesar de que tenga alguna *teja* que incumpla los requisitos de zinc, puede ser aprovechable por la empresa. Por ejemplo, la Figura 6.10 muestra los diferentes valores de zinc medidos en una bobina por cada sensor, junto con las líneas verdes entre las que se debería de encontrar la capa zinc. En dichas gráficas se pueden ver claramente como hay varias *tejas* que incumplen los requisitos (puntos que no se encuentran entre las dos líneas verdes). Pese a todo, la clase otorgada a esta bobina por parte del operario es OK.

### Tipos de datos

En primer lugar, hay que conocer los tipos de datos proporcionados, ya que según los sensores que los miden (se explicarán más adelante), los datos tomados de la bobina pueden ser 1D o 2D:

- **Datos 1D:** los datos se van tomando *teja* por *teja*, de forma que realmente se está construyendo un vector unidimensional. La Figura 6.5 lo muestra mucho más claro de forma gráfica, donde, en primer lugar, se miden los datos de la *teja* 1, a continuación los de la *teja* 2 y así sucesivamente hasta leer toda la bobina.
- **Datos 2D:** en este otro tipo de datos, en vez de utilizar un único sensor, se usan 9, de forma que sobre cada *teja* se toman 9 datos diferentes. La Figura 6.6 muestra como sería el proceso de lectura de datos. En primer lugar, se lee la primera *teja* y sobre ella se toman 9 valores (celdas 1 a la 9), a continuación se toman datos de la *teja* 2

1	2	3	4	5	6	7
---	---	---	---	---	---	---

Figura 6.5: Ejemplo estructura datos 1D

y se toman otros 9 datos (celdas 10 a 18). En definitiva, sobre cada bobina se está construyendo una matriz de 9 filas y un número de columnas cambiante en cada caso, según como de larga sea la bobina.

1	10	19	28	37	46	55	64	73
2	11	20	29	38	47	56	65	74
3	12	21	30	39	48	57	66	75
4	13	22	31	40	49	58	67	76
5	14	23	32	41	50	59	68	77
6	15	24	33	42	51	60	69	78
7	16	25	34	43	52	61	70	79
8	17	26	35	44	53	62	71	80
9	18	27	36	45	54	63	72	81

Figura 6.6: Ejemplo estructura datos 2D

### Tipos de sensores

Antes de empezar a explicar que atributo tiene cada tipo de dato, es importante conocer cómo funcionan los diferentes sensores, ya que cada uno de ellos opera de una forma diferente para medir los datos. En total existen 6 sensores diferentes, cada uno de ellos nombrado con una ID diferente:

- **123:** se corresponde con un sensor de datos 1D que mide la capa superior de la bobina. Toma datos cada 25 metros.
- **124:** se corresponde con la pareja del sensor 123, pero toma datos de la capa inferior de la bobina.

- **201:** se corresponde con otro sensor de datos 1D, pero diferente al anteriormente mencionado, de la capa superior. Este sensor también toma datos cada 25 metros.
- **202:** es la pareja del sensor 201. Toma medidas de la capa inferior de la bobina.
- **1234:** es un sensor de los datos 2D y toma datos de la capa superior de la bobina. Los datos son tomados cada 10 metros.
- **1243:** es la pareja del sensor 1234, y toma datos de la capa inferior de la bobina.

### Datos proporcionados

Los datos proporcionados se encuentran en una base de datos MySQL, y divididos en tres tablas. A continuación se recoge el nombre de cada una de ellas, junto con los principales atributos que serán utilizados en el proyecto:

- **GR\_LPValues:** contiene un total de 115972 registros de los datos 1D. Cada registro se corresponde para una bobina, sensor y *teja* específica. Los principales atributos son:
  - **COILID:** se corresponde con la ID de la bobina.
  - **TILEID:** se corresponde con la ID de la *teja* de la bobina. Cada *teja* tiene diferente ID, su orden es cronológico, es decir, la primera *teja* será el valor más pequeño, la segunda *teja*, el segundo valor más pequeño, y así sucesivamente.
  - **MID:** se corresponde con la ID del sensor que ha capturado los datos. Estos valores son los mencionados anteriormente, dónde se explicaba como funcionaba cada sensor.
  - **MEAN:** contiene el valor medio de zinc en dicha *teja*. Este es el valor utilizado para ver si cumple o no con los requisitos de zinc.

Un ejemplo de los diferentes atributos que componen esta tabla se pueden ver en la Figura 6.7.

- **GR\_QPValues:** contiene los registros de los datos 2D. En total hay 1272312 registros, y de igual manera que en los datos 1D, cada registro se corresponde para una bobina, sensor y *teja*. Los atributos usados son los mismos que los de la tabla anterior, pero con la diferencia que ahora el atributo TILEID contiene a la ID de cada celda, en vez de

	COILID	TILEID	MID	PLANT	SIDE	MEAN	MAX	MIN	COUNT
0	225216688	0.0	123.0	1066.0	T	88.94490	88.94490	88.94490	1.0
1	225216688	5120.0	123.0	1066.0	T	81.65457	81.65457	81.65457	1.0
2	225216688	10240.0	123.0	1066.0	T	94.79793	94.79793	94.79793	1.0
3	225216688	15360.0	123.0	1066.0	T	127.82180	127.82180	127.82180	1.0
4	225216688	20480.0	123.0	1066.0	T	125.88000	125.88000	125.88000	1.0

Figura 6.7: Ejemplo tabla GR\_LPValues

cada *teja*, dando lugar a la estructura mostrada anteriormente en la Figura 6.6. Además, la Figura 6.8 muestra un ejemplo de 5 registros pertenecientes a esta tabla.

	COILID	TILEID	MID	PLANT	SIDE	MEAN	MAX	MIN	COUNT
0	225216688	0.0	1234.0	1066.0	T	147.09500	147.09500	147.09500	1.0
1	225216688	28.0	1234.0	1066.0	T	143.53551	143.53551	143.53551	1.0
2	225216688	56.0	1234.0	1066.0	T	132.94450	132.94450	132.94450	1.0
3	225216688	85.0	1234.0	1066.0	T	141.23540	141.23540	141.23540	1.0
4	225216688	113.0	1234.0	1066.0	T	143.35880	143.35880	143.35880	1.0

Figura 6.8: Ejemplo tabla GR\_QPValues

- **V\_Coils:** contiene las bobinas (cuyas IDs se encuentran en el atributo SID) con la etiqueta colocada por parte del operario (atributo *CLASSLABEL*). Además, tiene los valores mínimos y máximos de zinc (atributos *ZnMin* y *ZnMax*) que tiene que tener cada bobina. En total hay 1160 registros, o lo que es igual, 1160 bobinas clasificadas por parte de un operario. De dichas 1160 bobinas, 325 pertenecen a la clase NOK, lo que supondría un 28 % de los datos, mientras que 835 bobinas pertenecen a la clase OK, suponiendo un 72 % de los datos. Tras esta distribución, se puede ver cómo el conjunto de datos está desbalanceado. Finalmente, la Figura 6.9 muestra un ejemplo de 5 registros correspondientes a esta tabla.

## Visualización de los datos

En última instancia, en esta segunda fase se decidió realizar una pequeña visualización de los datos 1D de las bobinas proporcionadas junto con dos líneas verdes, representando los valores máximos y mínimos de zinc. Para ello, se llevó a cabo un desplegable con las IDs de las diferentes bobinas,

	id	SID	PLANT	NAME	STARTTIME	MATERIAL	LENGTH	WIDTH	THICK	WEIGHT	BASEPID	CLASSLABEL	Label	ZnMin	ZnMax
0	1161	226031456	1066.0	226031456	2022-01-28	0.0	792.0	1183.0	3.0	22.0	220100.0	NOK	0	60	80
1	1162	226101621	1066.0	226101621	2022-01-31	0.0	273.0	1252.0	4.0	10.0	220100.0	NOK	2	215	258
2	1163	226120037	1066.0	226120037	2022-01-31	0.0	599.0	1305.0	2.0	15.0	220100.0	OK	1	155	175
3	1164	225387481	1066.0	225387481	2022-01-13	0.0	268.0	1502.0	5.0	15.0	220100.0	OK	1	60	80
4	1165	225406471	1066.0	225406471	2022-01-13	0.0	735.0	1158.0	2.0	19.0	220100.0	NOK	0	45	58

Figura 6.9: Ejemplo tabla V\_Coilss

de tal forma que al seleccionar una, se mostraban los gráficos de los cuatro sensores.

La Figura 6.10 muestra un ejemplo de visualización de las cuatro gráficas correspondientes a los diferentes sensores, junto con los diferentes valores de zinc medidos. En este caso se puede ver claramente, al observar el sensor 123, como la mayoría de las *tejas* de la capa superior no cumplen con los requisitos de zinc. En cambio, si miramos los otros 3 sensores, se puede ver cómo la mayoría de las *tejas* sí que cumple con los requisitos.

Bobina 225216688

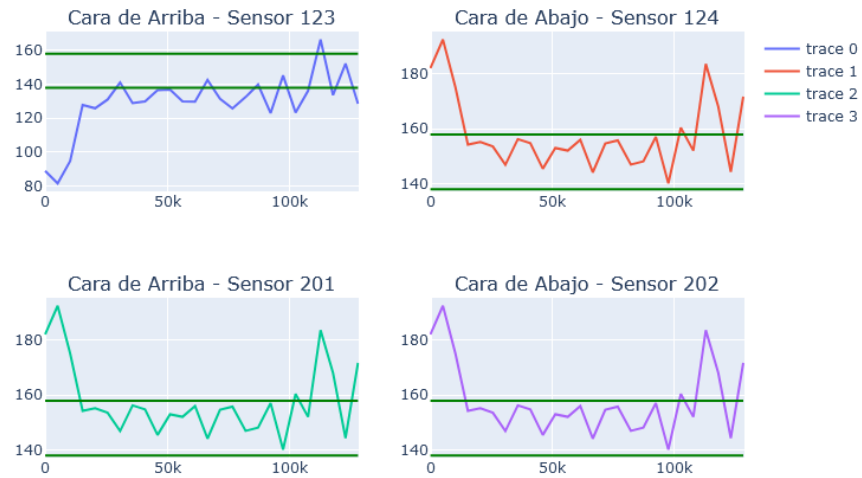


Figura 6.10: Ejemplo de Visualización para una Bobina

Este caso de ver como los valores medidos por el sensor 123 son bastante diferentes a los medidos 201 (ambos sensores miden la capa superior de la bobina), nos sorprendió, ya que los resultados deberían de ser similares al estar midiendo la misma capa, fue notificado a la empresa, aunque no se nos indicó nada al respecto.

## 6.3. Preparación de los datos

Esta tercera fase se ha dividido en dos grandes etapas: obtención de características y procesamiento de los datos para los modelos.

### Obtención de características

Las características obtenidas para cada bobina fueron las indicadas por la empresa, ya que se estimaba que iban a ser las que mejores resultados iban a dar.

Todas las características, que se van a mostrar a continuación, se han calculado para cada bobina y sensor y para los datos 1D y 2D. Es decir, para los datos 1D, se calcularon las características 4 veces, una para cada sensor. En cambio, para los datos 2D, al haber dos sensores, se calcularon dos veces.

- **ZNMAX\_FAILURES:** se corresponde al número de tejas de la bobina que tiene una capa de zinc superior al estipulado. Los fallos se contarán desde que hay un fallo hasta que se encuentra una teja correcta, no un fallo por cada teja con un defecto.
- **ZNMIN\_FAILURES:** es el número de tejas de la bobina que tiene menos zinc del mínimo estipulado. Los fallos también se cuentan con el mismo criterio que el caso anterior.
- **CALIBRATED:** se corresponde al número de tejas en las que se ha producido un calibrado (reinico de los sensores).
- **TOTAL\_TILEID:** indica el número total de las tejas que tiene la bobina.
- **L\_DIS:** contiene el número de tejas que hay desde el inicio de la bobina hasta el último fallo antes de la mitad de la bobina.
- **R\_DIS:** indica el número de tejas que hay desde el fallo más cercano a la mitad y el final de la bobina.

Finalmente, dentro de las características calculadas para cada bobina, además de las anteriormente descritas, se calculó el mapa codificado de cada bobina. Este mapa consiste en que a cada teja de la bobina se le asigna uno de los siguientes valores:

- **1:** en caso de que el valor de zinc medido en la teja sea mayor al estipulado.
- **0:** en caso de que el valor de zinc medido en la teja se encuentre entre el intervalo de zinc máximo y mínimo estipulado.
- **-1:** en caso de que el valor de zinc medido en la teja sea menor al estipulado.

Este mapa se almacenó en forma de *array*, incluido para los datos 2D, ya que pese a que su mapa codificado es una matriz, por comodidad se ha guardado de esta forma. Será a la hora de leerlo cuando se transforme en matriz para poder utilizarlo correctamente.

La Figura 6.11 muestra un ejemplo de salida para los datos 1D. En ella se pueden ver todos los atributos anteriormente mencionados para cada sensor, incluido el mapa codificado correspondiente a la columna MAP.

COILID	MID	ZNMAX_FAILURES	ZNMIN_FAILURES	CALIBRATED	TOTAL_TILEID	L_DIS	R_DIS	MAP	DECISION_OP
225216688	123.0	0	6	0	26	13	12	[-1, -1, -1, -1, -1, -1, 0, -1, -1, -1, -1, -1, ...]	OK
225216688	124.0	4	0	0	26	3	6	[1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]	OK
225216688	201.0	1	1	0	26	3	4	[-1, -1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]	OK
225216688	202.0	1	1	0	26	13	13	[1, 1, 1, -1, -1, -1, -1, -1, -1, -1, -1, -1, ...]	OK

Figura 6.11: Ejemplo de características obtenidas para datos 1D

Por otro lado, la Figura 6.12 muestra un ejemplo para la misma bobina que la anterior pero con datos 2D. De igual manera, se pueden ver todas las características anteriormente descritas.

COILID	MID	ZNMAX_FAILURES	ZNMIN_FAILURES	CALIBRATED	TOTAL_TILEID	L_DIS	R_DIS	MAP	DECISION_OP
225216688	1234.0	7	61	0	576	288	282	[0, 0, -1, 0, 0, -1, -1, -1, -1, 0, 0, -1, -1, ...]	OK
225216688	1243.0	56	46	0	576	288	286	[0, 0, -1, -1, 1, 0, 0, 1, 1, 0, 0, -1, -1, ...]	OK

Figura 6.12: Ejemplo de características obtenidas para datos 2D

Para finalizar esta etapa, se guardaron las características obtenidas de los datos 1D y 2D en una base de datos. Para ello se crearon dos tablas, FEATURES\_1D y FEATURES\_2D, donde se almacena todo el contenido previamente calculado.

## Preparación de los datos para los modelos

En esta segunda etapa de la tercera fase, era turno de preparar los datos obtenidos en la etapa anterior para que puedan ser utilizados por los modelos de *TensorFlow*.



En primer lugar, se cargaron los mapas codificados de las bobinas 1D, y se decidió unir en un único *array* cada par de sensores, es decir, unir los mapas de los sensores 123 y 124 y, por otro lado, los sensores 201 y 202.

El porqué de unirlos fue básicamente para tener los datos en una única dimensión, al igual que los originales, y así aplicar convoluciones de una dimensión.

A continuación, sobre estos nuevos mapas unidos, se aplicó la técnica de *padding*, que se basa en rellenar los datos con un valor para que todos ellos tengan las mismas dimensiones, ya que es necesario para las redes neuronales que se van a construir. En este caso, se rellenaron los datos con 0, hasta que todos los datos tuvieran una longitud de 208, puesto que fue el valor más grande encontrado dentro del conjunto de datos.

El porqué de añadir ceros se debe a que este valor representa que la teja es correcta, por tanto, si estamos añadiendo valores ficticios creíamos que era mejor añadir un valor correcto que no 1 o -1, indicando que la teja tiene algún defecto, y tal vez, afectando negativamente a la decisión del modelo.

En segundo lugar, se cargaron los mapas codificados de los datos 2D. Primero que todo, se reconstruyeron en forma de matriz, ya que como se ha comentado previamente, se guardaron en forma de *array* por comodidad. Sobre estas matrices, también se aplicó la técnica de *padding*, para que todos los mapas contaran con el mismo número de columnas.

Finalmente, con el resto de características, se decidió normalizar todas ellas, ya que de esta forma se contaba con valores para que los modelos trabajen. El tipo de normalización usada ha sido la conocida como normalización min-max, puesto que permite dejar los datos en valores comprendidos entre 0 y 1, y cuya fórmula es la siguiente:

$$DatoNorm = \frac{Dato - DatoMin}{DatoMax - DatoMin}$$

Ahora, ya se contaba con las características preparadas para ser usadas, pero antes había que transformar la clase de *string* a *int*. Dicha transformación fue: un 0 para la clase OK y un 1 para la clase NOK.

## 6.4. Modelado

En esta cuarta fase era turno de construir diversos modelos, con el fin de obtener el que mejor se adapte a este problema.

En esta fase, se pueden distinguir dos etapas diferentes. Por un lado, se realizaron diversas pruebas con *TensorFlow*, con el fin de generar diversos modelos y familiarizarse con su construcción.

Por otro lado, tenemos la segunda etapa, en ella se construyeron los diferentes modelos con varios parámetros para buscar el mejor de todos.

## Pruebas

En primer lugar, se crearon modelos individuales para familiarizarse con su construcción, la Figura 6.13, muestra un ejemplo de ello.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 206, 32)	128
max_pooling1d (MaxPooling1D)	(None, 103, 32)	0
flatten (Flatten)	(None, 3296)	0
dense (Dense)	(None, 32)	105504
dense_1 (Dense)	(None, 8)	264
dense_2 (Dense)	(None, 1)	9
Total params: 105,905		
Trainable params: 105,905		
Non-trainable params: 0		

Figura 6.13: Ejemplo de Construcción de Modelo

Pese a empezar con modelos individuales, se usará la estrategia de validación cruzada de 5 *folds*, ya que los resultados que se obtengan serán más robustos, al probar con diferentes conjuntos de valores para entrenar y validar los modelos.

Finalmente, en esta etapa también se decidió hacer pruebas utilizando alguna técnica para ajustar el conjunto de datos desbalanceado. En este caso se opta, debido al bajo número de registros, a realizar un sobremuestreo de los datos, y más concretamente aplicar la técnica SMOTE.

SMOTE (*Synthetic Minority Oversampling Technique*) se basa en producir más ejemplos de la clase minoritaria de manera sintética. Para ello, busca dos ejemplos que estén próximos y genera uno nuevo con datos intermedios

entre los dos ejemplos. Este proceso se repite varias veces hasta que el conjunto de datos se encuentra equilibrado.

Para concluir, indicar que los mejores resultados se consiguieron, aplicando validación cruzada, con datos a los que se les había aplicado el sobremuestreo, por lo que se decidirá usar en la siguiente etapa. Los resultados se pueden ver a continuación, en los que se muestran las matrices de confusión obtenidas para cada conjunto de datos. En las matrices, la primera fila se corresponde con la clase OK y la segunda fila con la clase NOK.

```
-- Datos originales--  
La matriz de confusion obtenida es:  
[[ 44   6]  
 [114  36]]  
  
-- Datos con sobremuestreo--  
La matriz de confusion obtenida es:  
[[40 10]  
 [94 56]]
```

Las precisiones obtenidas han sido de un 40 % y de un 48 % respectivamente, mostrándose unos mejores resultados en los datos con sobremuestreo.

### Construcción de modelos

A la hora de construir los modelos finales, se han probado dos estrategias. Por un lado, crear modelos que utilicen tan solo los mapas codificados de las bobinas, y, por otro lado, crear modelos que utilicen todas las características previamente calculadas. Además, los modelos serán diferentes, como es obvio, según si emplean datos 1D o 2D. Todo ello aplicando la validación cruzada de 5 *folds*.

Lo primero que se ha hecho en esta fase, ha sido separar una pequeña muestra de los datos originales: 150 registros de la clase NOK y 50 registros de la clase OK. De esta forma, se consigue que ningún modelo haya trabajado antes con estos datos y se puedan evaluar los modelos de una manera más exacta. Se ha separado esta muestra, ya que creemos que son datos suficientes como para evaluar y ver como se comportan los diferentes modelos. El resto de datos se usarán en la validación cruzada, de modo que en cada modelo algunos serán para entrenar y otros para validar.

En segundo lugar, se ha buscado el mejor criterio de discriminación para los modelos a construir. Para ello, se han probado un total de 5 valores diferentes y se ha obtenido el mejor de ellos, en función de las matrices de confusión y la precisión conseguida por cada criterio.

Los valores que se han probado han sido 0.3, 0.4, 0.5, 0.6, 0.7. Tras realizar diversas pruebas, se ha llegado a la conclusión que el mejor criterio es 0.3, tal y como se puede ver a continuación, donde se muestran las matrices de confusión y precisión obtenida (en las matrices de confusión la primera fila se corresponde a la clase Ok y la segunda a la clase NOK; con respecto a la precisión se corresponde a la tasa de acierto).

Los resultados del criterio 0.3 son:

La precisión obtenida ha sido 0.57  
La matriz de confusion obtenida es:  
[[33 17]  
[69 81]]

Los resultados del criterio 0.4 son:

La precisión obtenida ha sido 0.53  
La matriz de confusion obtenida es:  
[[36 14]  
[80 70]]

Los resultados del criterio 0.5 son:

La precisión obtenida ha sido 0.505  
La matriz de confusion obtenida es:  
[[40 10]  
[89 61]]

Los resultados del criterio 0.6 son:

La precisión obtenida ha sido 0.47  
La matriz de confusion obtenida es:  
[[40 10]  
[96 54]]

Los resultados del criterio 0.7 son:

```
La precisión obtenida ha sido 0.435
La matriz de confusion obtenida es:
[[ 42   8]
 [105  45]]
```

Tras realizar estos dos pasos, ha sido turno de construir los modelos. Como ya se ha comentado previamente, se ha utilizado validación cruzada para generar 5 modelos en cada experimento. En todos estos modelos la estrategia utilizada ha sido la siguiente:

- **Capa convolucional:** esta capa o capas, según las que se le indiquen, junto con el *kernel* indicado y su activación tipo *ReLU* (*Rectified Linear Unit*), aplican la técnica de convolución a los datos de entrada.
- **Pooling:** aplica la técnica *pooling* y devuelve el valor máximo.
- **Flatten:** transforma los datos de entrada en un vector unidimensional.
- **Dense:** capa o capas que contienen el número de neuronas indicado junto con una activación tipo *ReLU*.

La activación ReLU, anteriormente mencionada, es una función que añade, entre otras mejoras, la no linealidad a la red neuronal al transformar todos los números negativos en cero.

Además, a la hora de entrenar el modelo, se ha utilizado un *checkpoint*, que permite guardar el mejor modelo que se va obteniendo en las diferentes épocas, indicadas en el momento de su construcción, e ir sobrescribiéndole según se encuentra uno mejor, para quedarse con el modelo que menor *loss* tenga. La función *loss* permite evaluar el error obtenido entre las predicciones y los valores reales, de forma que cuanto menor sea mejor serán los resultados.

El tipo de *loss* empleado ha sido *binary cross entropy*, ya que suele ser la utilizada en problemas de clasificación binaria, cómo es el caso.

Con respecto a las predicciones finales de cada conjunto de modelos, se han sumado las predicciones de los 5 modelos y dividido entre 5, es decir, se ha calculado la media de los 5 modelos. A continuación, el valor se ha transformado a una clase u otra en función del criterio de discriminación, que, como ya se ha comentado anteriormente, se ha utilizado 0.3.

Finalmente, el número total de experimentos realizados ha sido 28. Para los datos 1D se han llevado a cabo 14 de ellos, de los cuales 7 han sido

utilizando como dato de entrada el mapa codificado y otros 7 con todas las características normalizadas obtenidas de las bobinas.

Para los datos 2D se han efectuado los otros 14 experimentos restantes, con la misma distribución que en los datos 1D.

Además, en los experimentos que utilicen todos las características normalizadas obtenidas de las bobinas, se ha probado a realizar, para cada experimento, tras obtener los 5 modelos, crear un *random forest* que con las 5 predicciones y los datos normalizados pretenda predecir la clase con el fin de saber si muestra mejores resultados. En resumen, los pasos que se han seguido para construir los *random forest* son los siguientes:

1. Con los datos de entrenamiento y test se obtienen los modelos con la validación cruzada.
2. Sobre los modelos obtenidos se hacen predicciones de los datos de entrenamiento y test.
3. Se unen las características normalizadas y las 5 predicciones de los modelos.
4. Se construye un *random forest* con los datos unidos y la clase real de los datos.
5. Se efectúan predicciones con los datos de la muestra y se comparan con los resultados obtenidos de emplear únicamente los 5 modelos, con el fin de identificar el que mejores resultados da.

A continuación se recogen los resultados obtenidos en los experimentos tanto para los datos 1D como para los datos 2D. En todas las matrices de confusión que se van a mostrar, la primera fila se corresponde a la clase OK, mientras que la segunda fila se corresponde a la clase NOK. Además, las precisiones vienen dadas en decimal, las cuales se corresponden a la tasa de acierto. Finalmente, añadir que las siglas RF se corresponden a los resultados del *random forest*.

También, aclarar que, en los resultados, se muestra tan solo la matriz de confusión obtenida al aplicar los 5 modelos obtenidos en cada experimento al aplicar validación cruzada, y no la matriz de confusión individual de cada modelo, ya que los resultados importantes son aquellos en los que se utilizan los 5 modelos.

## Resultados datos 1D

Se han realizado un total de 7 experimentos para los datos 1D sobre el mapa codificado de la bobina. Los resultados han sido los siguientes:

- **Experimento 1:** en este caso los parámetros utilizados han sido una capa convolucional con 32 filtros , el tamaño de *kernel* es 3, 300 iteraciones, y tres capas densas de tamaños, respectivamente, 32, 8 y 1. Los resultados obtenidos son los siguientes:

La precisión obtenida ha sido 0.535

La matriz de confusión obtenida se puede ver en la Figura 6.14.

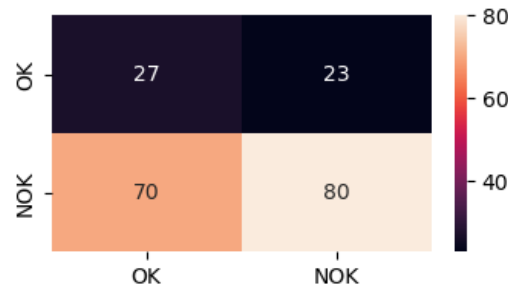


Figura 6.14: Matriz de Confusión Experimento 1

- **Experimento 2:** en este segundo experimento se han empleado una capa convolucional con 64 filtros y con un *kernel* de tamaño 5. Las iteraciones han sido 500, y las capas densas se han mantenido iguales que en el caso anterior. Los resultados son los siguientes:

La precisión obtenida ha sido 0.54

La matriz de confusión obtenida se puede ver en la Figura 6.15.

- **Experimento 3:** en este tercer caso se ha usado una capa convolucional con 16 filtros y con un *kernel* de tamaño 3. Las iteraciones han sido 300, y se han usado 5 capas densas de tamaños 64, 32, 16, 8 y 1. El resultado obtenido ha sido:

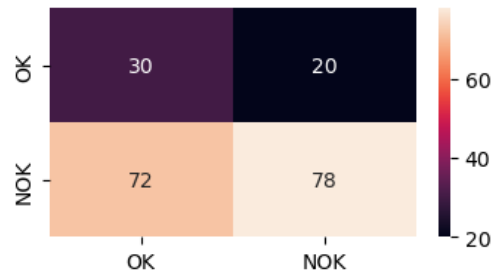


Figura 6.15: Matriz de Confusión Experimento 2

La precisión obtenida ha sido 0.54

La matriz de confusión obtenida se puede ver en la Figura 6.16.

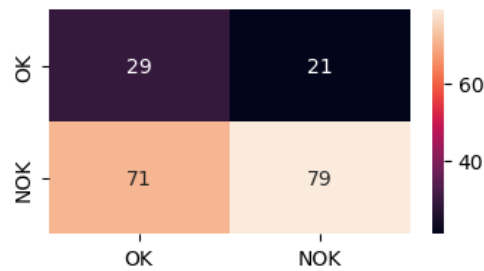


Figura 6.16: Matriz de Confusión Experimento 3

- **Experimento 4:** en este cuarto caso se ha usado una capa convolucional de 16 filtros y con un *kernel* de tamaño 5. Las iteraciones han sido 300 y se han empleado 3 capas densas de tamaño 32, 8 y 1. Los resultados obtenidos han sido los siguientes:

La precisión obtenida ha sido 0.56

La matriz de confusión obtenida se puede ver en la Figura 6.17.

- **Experimento 5:** en este quinto experimento se han usado 3 capas convolucionales con 16, 32 y 64 filtros y todas ellas con un *kernel* de tamaño 3. Las iteraciones y las capas densas han sido las mismas que las del experimento anterior. Los resultados obtenidos han sido;



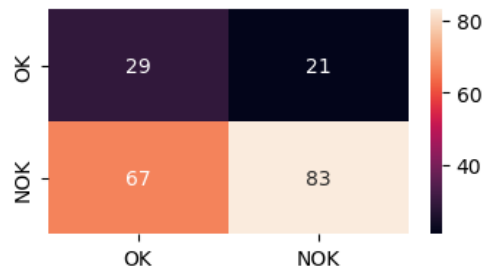


Figura 6.17: Matriz de Confusión Experimento 4

La precisión obtenida ha sido 0.57

La matriz de confusión obtenida se puede ver en la Figura 6.18.

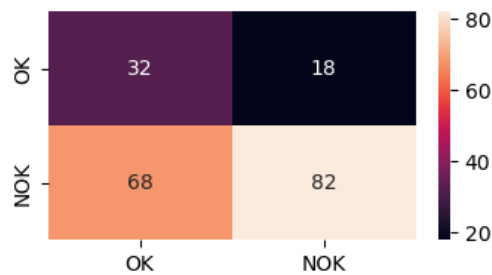


Figura 6.18: Matriz de Confusión Experimento 5

- **Experimento 6:** en este penúltimo experimento se han mantenido los valores del experimento anterior pero aumentando las iteraciones a 500. Los resultados han sido:

La precisión obtenida ha sido 0.56

La matriz de confusión obtenida se puede ver en la Figura 6.19.

- **Experimento 7:** en el último caso se han utilizado los mismos valores que en el experimento 5 pero cambiando el tamaño del *kernel* a 5. Los resultados han sido:

La precisión obtenida ha sido 0.505

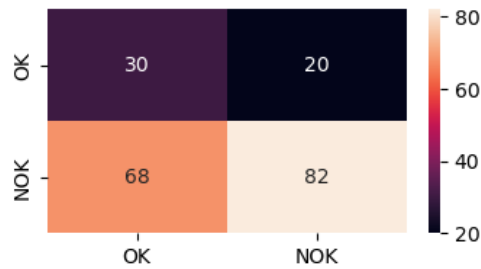


Figura 6.19: Matriz de Confusión Experimento 6

La matriz de confusión obtenida se puede ver en la Figura 6.20.

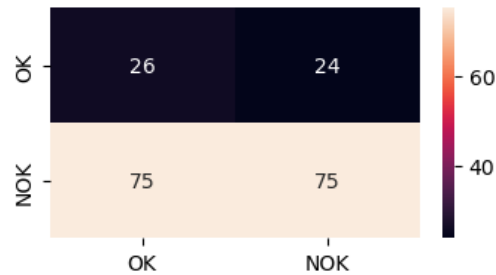


Figura 6.20: Matriz de Confusión Experimento 7

Sobre el mapa codificado y los atributos normalizados se han realizado otros 7 experimentos, cuyos resultados son los siguientes (en los resultados se incluye la precisión utilizando únicamente el modelo y la precisión obtenida combinando los modelos con el *Random Forest*):

- **Experimento 8:** este primer experimento ha sido con una capa convolucional con 32 filtros y con un *kernel* de tamaño 3. Además, tenía 300 iteraciones y 3 capas densas de tamaños 32, 8 y 1. Los resultados han sido:

La precisión obtenida ha sido 0.47

La matriz de confusión obtenida se puede ver en la Figura 6.21a.

La precisión obtenida con el RF ha sido 0.475

La matriz de confusión obtenida se puede ver en la Figura 6.21b.

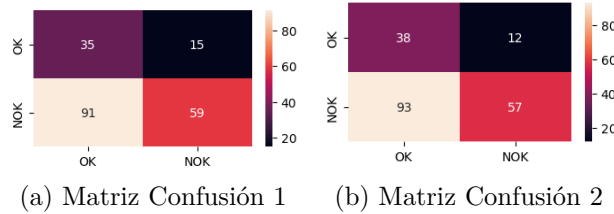


Figura 6.21: Matriz de Confusión Experimento 8

- **Experimento 9:** este segundo caso ha sido igual que el anterior pero con un *kernel* de tamaño 5 y con 16 filtros. Los resultados se muestran a continuación:

La precisión obtenida ha sido 0.51

La matriz de confusión obtenida se puede ver en la Figura 6.22a.

La precisión obtenida con el RF ha sido 0.445

La matriz de confusión obtenida se puede ver en la Figura 6.22b.

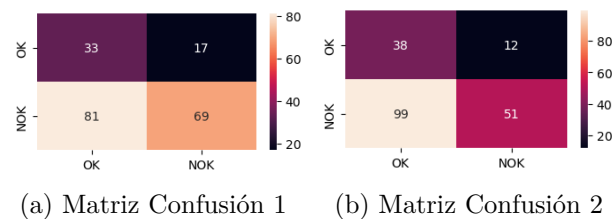


Figura 6.22: Matriz de Confusión Experimento 9

- **Experimento 10:** en este tercer experimento ha sido similar al anterior pero con 64 filtros. Los resultados han sido:

La precisión obtenida ha sido 0.485

La matriz de confusión obtenida se puede ver en la Figura 6.23a.

La precisión obtenida con el RF ha sido 0.46

La matriz de confusión obtenida se puede ver en la Figura 6.23b.

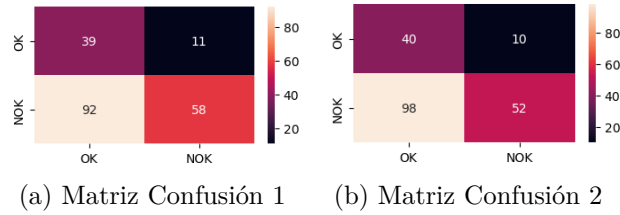


Figura 6.23: Matriz de Confusión Experimento 10

- **Experimento 11:** este cuarto caso ha sido igual que el primero pero con 16 filtros. Los resultados son:

La precisión obtenida ha sido 0.525

La matriz de confusión obtenida se puede ver en la Figura 6.24a.

La precisión obtenida con el RF ha sido 0.47

La matriz de confusión obtenida se puede ver en la Figura 6.24b.

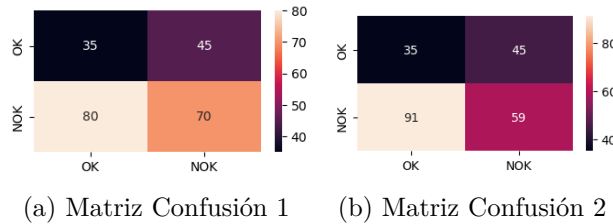


Figura 6.24: Matriz de Confusión Experimento 11

- **Experimento 12:** este quinto experimento ha sido igual que el anterior, solo que con 5 capas densas y de tamaños 64, 32, 16, 8 y 1. Se han obtenido los siguientes resultados:

La precisión obtenida ha sido 0.475

La matriz de confusión obtenida se puede ver en la Figura 6.25a.

La precisión obtenida con el RF ha sido 0.455

La matriz de confusión obtenida se puede ver en la Figura 6.25b.

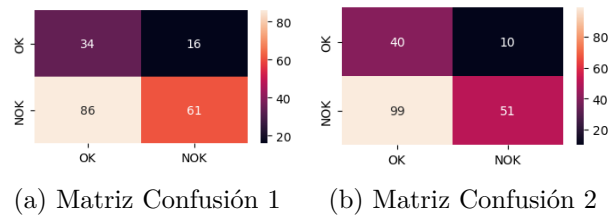


Figura 6.25: Matriz de Confusión Experimento 12

- **Experimento 13:** este penúltimo experimento ha sido igual que el anterior, pero con dos capas convolucionales con 32 y 64 filtros. Los resultados son:

La precisión obtenida ha sido 0.48

La matriz de confusión obtenida se puede ver en la Figura 6.26a.

La precisión obtenida con el RF ha sido 0.455

La matriz de confusión obtenida se puede ver en la Figura 6.26b.

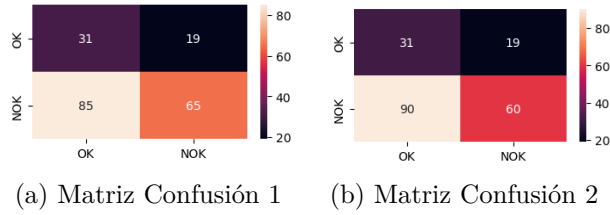


Figura 6.26: Matriz de Confusión Experimento 13

- **Experimento 14:** en el último caso, muy parecido al anterior, solo que con un *kernel* de tamaño 5, se han obtenido los siguientes resultados:

La precisión obtenida ha sido 0.425

La matriz de confusión obtenida se puede ver en la Figura 6.27a.

La precisión obtenida con el RF ha sido 0.435

La matriz de confusión obtenida se puede ver en la Figura 6.27b.

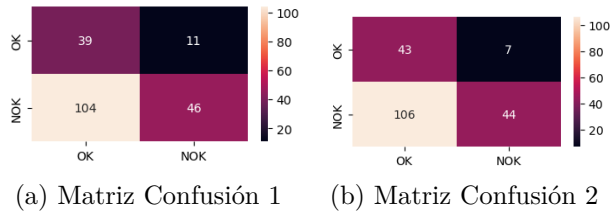


Figura 6.27: Matriz de Confusión Experimento 14

## Resultados datos 2D

Con respecto a los datos 2D, también se han realizado 7 experimentos sobre los mapas codificados de las bobinas. A continuación se muestran los resultados:

- **Experimento 15:** en este primer caso se utiliza una capa convolucional con 32 filtros y un *kernel* de tamaño 3x3. Además, se efectúan 100 iteraciones y tres capas densas de tamaños 32, 8 y 1. Los resultados han sido los siguientes:

La precisión obtenida ha sido 0.485

La matriz de confusión obtenida se puede ver en la Figura 6.28.

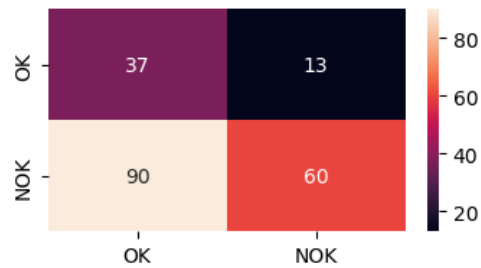


Figura 6.28: Matriz de Confusión Experimento 15

- **Experimento 16:** este experimento es muy similar al anterior, solo que con 16 filtros. Los resultados son los siguientes:

La precisión obtenida ha sido 0.49

La matriz de confusión obtenida se puede ver en la Figura 6.29.

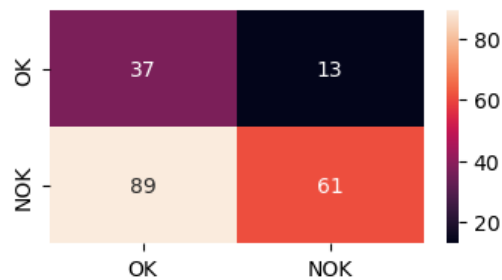


Figura 6.29: Matriz de Confusión Experimento 16

- **Experimento 17:** este tercer caso es muy similar al primero, pero ahora el tamaño del *kernel* es de 5x5. Sus resultados han sido:

La precisión obtenida ha sido 0.525

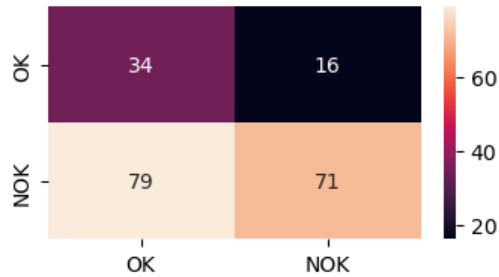


Figura 6.30: Matriz de Confusión Experimento 17

La matriz de confusión obtenida se puede ver en la Figura 6.30.

- **Experimento 18:** este tercer experimento es similar al primero, pero con 64 filtros. Además, tiene 5 capas densas de tamaños 64, 32, 16, 8 y 1. Sus resultados han sido:

La precisión obtenida ha sido 0.53

La matriz de confusión obtenida se puede ver en la Figura 6.31.

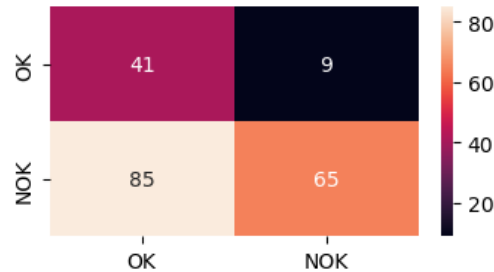


Figura 6.31: Matriz de Confusión Experimento 18

- **Experimento 19:** este experimento ha sido similar al primero, solo que con 16 filtros y un *kernel* de tamaño 5x5. Los resultados han sido:

La precisión obtenida ha sido 0.495

La matriz de confusión obtenida se puede ver en la Figura 6.32.



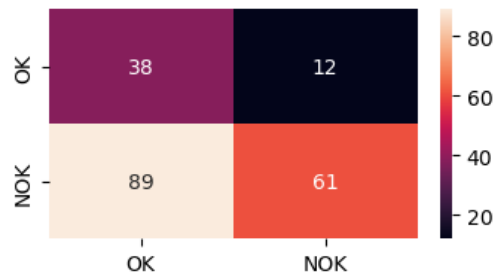


Figura 6.32: Matriz de Confusión Experimento 19

- **Experimento 20:** este caso ha sido similar al primero pero con dos capas convolucionales con 16 y 32 filtros. Sus resultados han sido:

La precisión obtenida ha sido 0.49

La matriz de confusión obtenida se puede ver en la Figura 6.33.

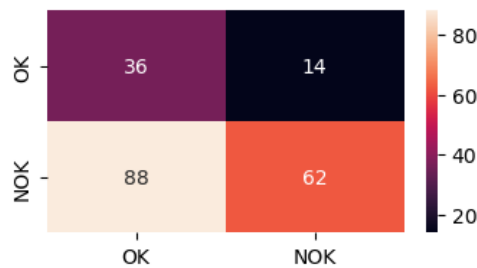


Figura 6.33: Matriz de Confusión Experimento 20

- **Experimento 21:** este experimento ha sido similar al anterior, pero en este caso con tres capas convolucionales, de tamaños 16, 32 y 64. Los resultados han sido:

La precisión obtenida ha sido 0.54

La matriz de confusión obtenida se puede ver en la Figura 6.34.

Para los datos 2D juntando el mapa con los demás atributos, se han realizado de igual manera 7 experimentos. Además, al igual que para los

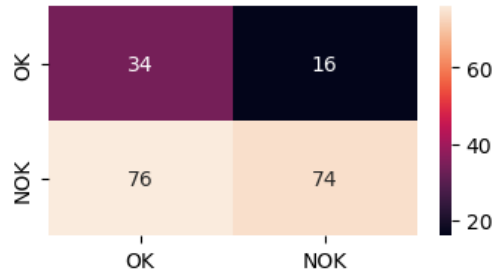


Figura 6.34: Matriz de Confusión Experimento 21

datos 1D, se han añadido los resultados de usar *Random Forest*. Todos los resultados se pueden ver a continuación:

- **Experimento 22:** en este primer experimento se utiliza una capa convolucional con 32 filtros y un *kernel* de 3x3. Además, se han empleado 100 iteraciones y 3 capas densas de tamaño 32, 8 y 1. Sus resultados han sido:

La precisión obtenida ha sido 0.51

La matriz de confusión obtenida se puede ver en la Figura 6.35a.

La precisión obtenida con el RF ha sido 0.5

La matriz de confusión obtenida se puede ver en la Figura 6.35b.

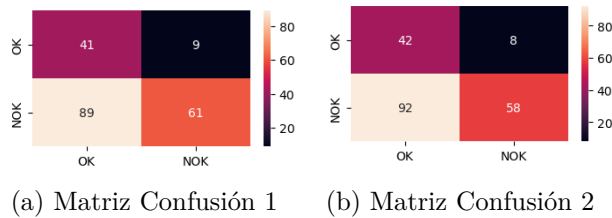


Figura 6.35: Matriz de Confusión Experimento 22

- **Experimento 23:** este segundo experimento ha sido similar al anterior, pero con 16 filtros y el *kernel* de tamaño 5x5. Los resultados han sido:

La precisión obtenida ha sido 0.505

La matriz de confusión obtenida se puede ver en la Figura 6.36a.

La precisión obtenida con el RF ha sido 0.53

La matriz de confusión obtenida se puede ver en la Figura 6.36b.

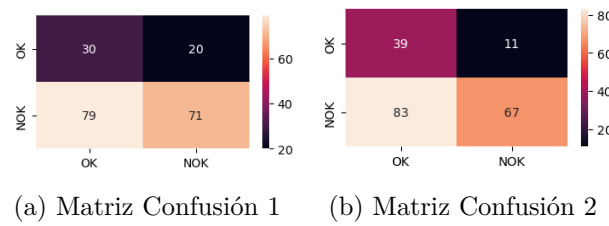


Figura 6.36: Matriz de Confusión Experimento 23

- **Experimento 24:** este caso ha sido muy similar al primero pero con un *kernel* de 5x5. Los resultados han sido:

La precisión obtenida ha sido 0.525

La matriz de confusión obtenida se puede ver en la Figura 6.37a.

La precisión obtenida con el RF ha sido 0.51

La matriz de confusión obtenida se puede ver en la Figura 6.37b.

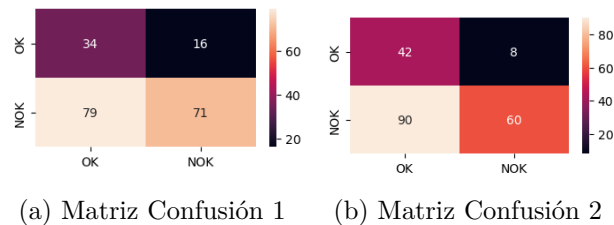


Figura 6.37: Matriz de Confusión Experimento 24

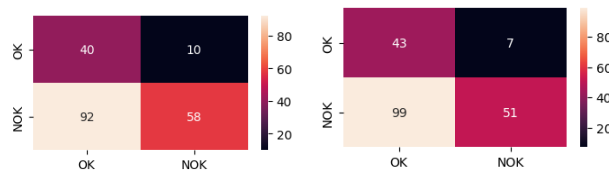
- **Experimento 25:** este experimento ha sido como el primero, pero esta vez utilizando 16 filtros. Sus resultados han sido:

La precisión obtenida ha sido 0.49

La matriz de confusión obtenida se puede ver en la Figura 6.38a.

La precisión obtenida con el RF ha sido 0.47

La matriz de confusión obtenida se puede ver en la Figura 6.38b.



(a) Matriz Confusión 1      (b) Matriz Confusión 2

Figura 6.38: Matriz de Confusión Experimento 25

- **Experimento 26:** este experimento ha sido similar al primero, pero las capas densas pasan a ser 5 y sus tamaños son 64, 32, 16, 8 y 1. Sus resultados son:

La precisión obtenida ha sido 0.515

La matriz de confusión obtenida se puede ver en la Figura 6.39a.

La precisión obtenida con el RF ha sido 0.495

La matriz de confusión obtenida se puede ver en la Figura 6.39b.

- **Experimento 27:** en este sexto caso, similar al primero, se han añadido dos capas convolucionales con 32 y 16 filtros. Los resultados son:

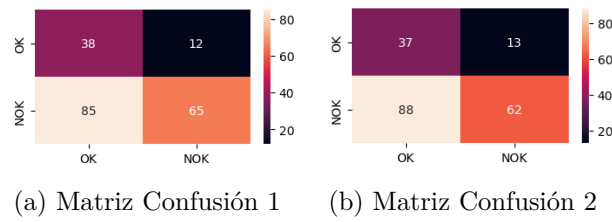


Figura 6.39: Matriz de Confusión Experimento 26

La precisión obtenida ha sido 0.51

La matriz de confusión obtenida se puede ver en la Figura 6.40a.

La precisión obtenida con el RF ha sido 0.515

La matriz de confusión obtenida se puede ver en la Figura 6.40b.

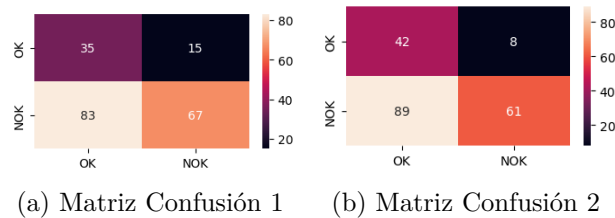


Figura 6.40: Matriz de Confusión Experimento 27

- **Experimento 28:** este experimento ha sido similar al anterior, pero con una capa convolucional más. Sus tamaños son 16, 32 y 64, y cuyos resultados son:

La precisión obtenida ha sido 0.49

La matriz de confusión obtenida se puede ver en la Figura 6.41a.

La precisión obtenida con el RF ha sido 0.475

La matriz de confusión obtenida se puede ver en la Figura 6.41b.

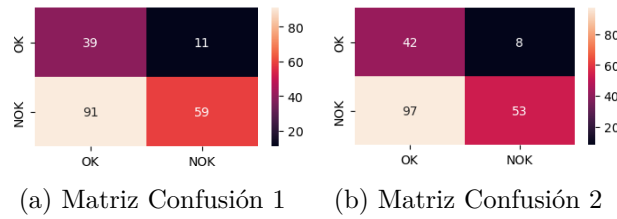


Figura 6.41: Matriz de Confusión Experimento 28

## 6.5. Evaluación

En esta quinta y penúltima fase, se han analizado los resultados obtenidos en los 28 experimentos realizados en la fase anterior.

Lo primero que resalta son los malos resultados, ya que ninguno de los 28 experimentos han mostrado una buena precisión, de hecho, casi todos están con una precisión de entre el 50 % y 60 %.

Si comparamos los resultados entre los datos 1D y 2D, se ve como las precisiones más altas son las del primer tipo de datos, algo que puede resultar extraño, ya que se puede presuponer que los datos 2D mostrarán algún patrón o característica que haga que el modelo sea mejor, pero no ha sido el caso. Es por ello, que se cree que esto se debe a que en los datos proporcionados, el operario usa para clasificar las bobinas los datos 1D, de ahí esos posibles mejores resultados.

Si comparamos los resultados obtenidos directamente con los modelos a los obtenidos por el *random forest*, se ve como este último, casi siempre es similar o un poco peor que si utilizamos solo los modelos.

Finalmente, el mejor resultado, se ha obtenido con el experimento 5 de los datos 1D empleando tan solo el mapa codificado de la bobina, por lo que será este el que se empleará en la siguiente fase.

## 6.6. Despliegue

En esta sexta y última fase, ha sido turno de construir la aplicación final del proyecto, sobre la cual se puedan cargar datos de bobinas medidos por sensores y predecir y si serán bobinas válidas o no.

La construcción de la aplicación se ha basado en dos partes. Por un lado, tenemos un *notebook* donde se podrá elegir la base de datos donde se encuentran los datos y visualizar los resultados. Y, por otro lado, tenemos un archivo de *Python* que contiene las diferentes funciones y que encapsula, de cara al usuario, todo el proceso, con el fin de dejar la aplicación lo más sencilla y visual posible.

Con respecto a los modelos usados para las predicciones, se han correspondido a los del experimento 5 sobre los datos 1D, ya que, como ya se ha visto en la fase anterior, han sido los que mejores resultados han dado.

Además, para añadirle más funcionalidad a la aplicación, cada vez que se realizan predicciones sobre un modelo, se genera un archivo CSV que contiene todas las características obtenidas de la bobina, puesto que puede ser interesante almacenar los valores por si fuesen de utilidad al operario. También, se ha creado un histórico donde se almacenan las predicciones de las bobinas para poder consultarlas en el futuro sin la necesidad de tener que volver a cargar los datos.

El resultado se puede ver en la Figura 6.42, donde se aprecia la tabla con las diferentes características calculadas y el mapa codificado de la bobina. Además, más abajo se puede ver la predicción obtenida por los modelos según los datos obtenidos por cada par de sensores.

-----

Bobina seleccionada: 226861024

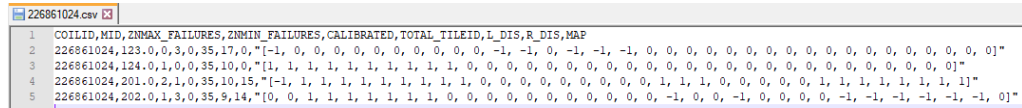
	COILID	MID	ZNMAX_FAILURES	ZNMIN_FAILURES	CALIBRATED	TOTAL_TILEID	L_DIS	R_DIS	MAP
0	226861024	123.0	0	3	0	35	17	0	[-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 0, ...
1	226861024	124.0	1	0	0	35	10	0	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, ...
2	226861024	201.0	2	1	0	35	10	15	[-1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, ...
3	226861024	202.0	1	3	0	35	9	14	[0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, ...

Predicción según los sensores: 123 y 124  
La predicción de la bobina es la clase OK

Predicción según los sensores: 201 y 202  
La predicción de la bobina es la clase OK

Figura 6.42: Ejemplo de Resultado de la Aplicación

Y como se ha comentado, se ha generado a su vez un archivo CSV con todos los atributos calculados de la bobina y su mapa codificado para cada sensor. La Figura 6.43, muestra el CSV generado para la bobina de ejemplo mostrada en la imagen anterior.



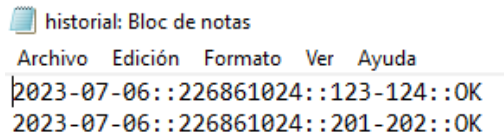
```

1 COILID,MID,ZNMAX_FAILURES,ZNMIN_FAILURES,CALIBRATED,TOTAL_TILEID,L_DIS,R_DIS,MAP
2 226861024,123,0,0,35,17,0,"[-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 0, -1, -1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]"
3 226861024,124,0,1,0,35,10,0,"[1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]"
4 226861024,201,0,2,1,0,35,10,15,"[-1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]"
5 226861024,202,0,1,3,0,35,9,14,"[0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]"

```

Figura 6.43: Ejemplo de Resultado del CSV Genrado por la Aplicación

Y también se ha generado un archivo historial.txt que contiene el registro de predicciones. La Figura 6.44, muestra dicho resultado.



```

historial: Bloc de notas
Archivo Edición Formato Ver Ayuda
2023-07-06::226861024::123-124::OK
2023-07-06::226861024::201-202::OK

```

Figura 6.44: Ejemplo de Resultado del historial.txt Genrado por la Aplicación

Además, si siguiéramos realizando predicciones, se generaría un CSV correspondiente a la bobina y en el fichero correspondiente al historial se añadirían al final la nueva información.



---

# Conclusiones y Líneas de trabajo futuras

---

## 7.1. Conclusiones

En primer lugar, comentar que se han cumplido todos los objetivos generales marcados al inicio del proyecto. A continuación se recogen los diferentes objetivos junto con la explicación de porque se ha cumplido:

- **Crear atributos que describan la calidad del producto y puedan ser utilizados posteriormente como atributos para modelar:** se han modelado diferentes atributos de las bobinas que posteriormente han sido utilizados en un total de 28 experimentos diferentes, en los que se han buscado los mejores modelos que se adaptan al problema del proyecto y a los atributos calculados.
- **Evaluar diferentes estrategias de modelado con técnicas de *machine learning* y *deep learning* para realizar recomendaciones:** dentro de los 28 experimentos se han realizado 14 para datos 1D y otros 14 experimentos para 2D. En dichos experimentos, según el tipo de datos, se han probado diversas estrategias buscando la mejor de ellas.
- **Realizar una aplicación utilizable por un operario para validar diferentes bobinas:** el proyecto va acompañado de un *notebook* que representa la aplicación del proyecto y que simula el proceso que un operario seguiría para llevar a cabo las predicciones sobre unas bobinas en concreto.

Pese a estar contentos por cumplir con todos los objetivos generales marcados, no se han obtenido unas precisiones tan buenas como las deseadas. Es por ello, que creemos que al contar con un conjunto de datos desbalanceado y con, tal vez, pocos ejemplos, los resultados no han sido tan buenos como los que cabría esperar. A pesar de ello, pensamos que se han seguido buenas estrategias frente a los datos prestados.

Por otro lado, con los 28 experimentos realizados, se puede sacar en claro que los que mejores resultados han dado, han sido en los casos en los que se han empleado datos 1D de las bobinas, y más en concreto aquellos que utilizan el mapa codificado de las bobinas.

En resumen, se está contento con las diferentes estrategias marcadas y seguidas a lo largo del proyecto, junto con la multitud de diferentes pruebas que se han llevado a cabo. Y aunque los resultados no sean los esperados, se han aprendido multitud de lecciones y conocido un nuevo ámbito laboral relacionado con el máster.

## 7.2. Líneas de trabajo futuras

Bajo mi punto de vista, creo que los siguientes pasos y mejoras del proyecto son:

1. **Aumento de bobinas clasificadas pertenecientes a la clase NOK:** con el fin de tener un conjunto de datos más equilibrado y poder crear mejores modelos, sería interesante obtener más bobinas pertenecientes a la clase NOK. De igual manera, si el número de bobinas pertenecientes a la clase OK también aumenta, es bastante posible que el modelo obtenido sea más preciso. En definitiva, cuantos más datos se posean y más equitativo sea el conjunto de datos, los resultados que se consigan serán seguramente mejores.
2. **Probar otras alternativas:** es posible que si se emplea algún tipo de red neuronal diferente al empleado en el proyecto, o que si se obtienen nuevas características de las bobinas, los modelos muestren mejores resultados.
3. **Desarrollar una aplicación web:** la aplicación actual es un *notebook* sobre el cual se pueden cargar bobinas y con los modelos generados realizar predicciones sobre si serán válidas o no. Pero su interfaz no es del todo amigable, sobre todo para gente que no haya hecho nunca programación y es necesario tener en ejecución el archivo para poder

utilizarlo. Es por ello, que sería una gran mejora hacer la aplicación en un entorno web con una interfaz sencilla para que los operarios puedan emplearla fácilmente.



# Apéndice



## Apéndice A

---

# Plan de Proyecto Software

---

### A.1. Introducción

Este primer apéndice contiene información relacionada con la planificación temporal llevada a cabo durante el proyecto y su diferentes viabilidades.

### A.2. Planificación temporal

Como ya se ha comentado previamente en la memoria, al hablar de la metodología seguida, la planificación del proyecto se ha basado en la metodología de trabajo CRISP-DM. Las reuniones se producían, salvo alguna excepción, cada dos semanas, donde se revisaban los objetivos marcados en la semana anterior y se marcaban los objetivos de las próximas dos semanas.

Además, para la gestión del proyecto y poder reflejar el desarrollo iterativo se ha empleado *GitHub*. El repositorio del proyecto para poder observarlo es el siguiente: <https://github.com/ifah1001/TFM>.

### A.3. Estudio de viabilidad

Con respecto al estudio de la viabilidad, se va a proceder a analizar la viabilidad económica y legal del proyecto.

#### Viabilidad económica

Dentro de la viabilidad económica, podemos encontrar dos tipos de gastos:

- **Coste *hardware*:** correspondiente al soporte físico necesario para mantener activa la aplicación.
- **Coste personal:** correspondiente a los contratos de las diferentes personas necesarias para mantener activa la aplicación.

### Coste *hardware*

La aplicación para poder ejecutarse necesita estar lanzada a través de *Jupyter Notebook*, por lo que los costes necesarios se pueden ver en la Tabla A.1. En ella se incluyen los elementos básicos necesarios para que un operario pueda utilizar la aplicación.

Elemento	Coste en €
Ordenador	750
Monitor	100
Teclado y ratón	20
<b>Coste total</b>	<b>870</b>

Tabla A.1: Costes *hardware* estimados

### Coste personal

Para calcular este tipo de gasto, se supone que se contrata a un programador y que en España tienen un salario medio de 29.800€ brutos al año (cantidad obtenida de Jobted<sup>1</sup>). En la Tabla A.2 se puede ver el diferente desglose de gastos según el sueldo bruto estipulado.

Concepto	Coste en €
Sueldo neto anual	23.081,5
Cuota a pagar en el IRPF (16,20 %)	4.826,2
Cuotas a la Seguridad Social	1.892,3
<b>Sueldo bruto anual</b>	<b>29.800</b>

Tabla A.2: Costes personal estimados

### Coste total

Tras haber analizado ambos tipos de costes, en la Tabla A.3 se muestran los costes anualizados para contratar a un programador junto con los costes de *hardware* necesarios.

<sup>1</sup>Jobted: <https://www.jobted.es/salario/programador>



Tipo de coste	Coste en €
Coste hardware	870
Coste personal	29.800
<b>Coste total</b>	<b>30.670</b>

Tabla A.3: Costes totales estimados

## Viabilidad legal

El proyecto se ha llevado a cabo en *Python* con diferentes librerías gratuitas, y que todas ellas permiten su comercialización gratuita.

El problema es que para construir los diferentes modelos en el proyecto, se han empleado datos de bobinas reales y medidas por una empresa, que cómo ya se ha comentado previamente, no tenemos su permiso para mencionar su nombre. Es por ello, que los datos empleados han sido tomados con unos sensores específicos, y si alguien desea utilizar la aplicación, necesita tener datos medidos por sensores similares a los de la empresa, algo que puede suponer un inconveniente.

Tras todo lo anteriormente comentado, se ha decidido analizar las diferentes licencias asociadas para cada biblioteca y herramienta empleada, y decidir la licencia final del proyecto. Todo esto se puede ver en la Tabla A.4.

Bibliotecas y Herramientas	Licencia
Jupyter Notebook	BSD
Pandas	BSD
NumPy	BSD
MySQL	GNU
PyMySQL	MIT
TensorFlow	Apache 2.0
Scikit-Image	BSD
Imbalanced-Learn	MIT
Matplotlib	PSF
Jupyter Widgets	BSD

Tabla A.4: Licencias Bibliotecas Python y Herramientas

Tras analizar las diferentes licencias de las diferentes bibliotecas y herramientas utilizadas, he optado por colocar al proyecto una licencia GPL v3, ya que de esta forma se puede modificar, distribuir, comercializar y realizar patentes del *software* generado durante el desarrollo del proyecto.



## Apéndice *B*

---

# Especificación de Requisitos

---

### B.1. Introducción

Este segundo apéndice recuerda los objetivos generales marcados en el proyecto, y que previamente se han comentado en la memoria. Además tratará el catálogo de requisitos junto con su especificación y diagramas de caso de uso.

### B.2. Objetivos generales

El proyecto se ha marcado los siguientes objetivos generales:

- Crear atributos que describan la calidad del producto y puedan ser utilizados posteriormente como atributos para modelar.
- Evaluar diferentes estrategias de modelado con técnicas de *machine learning* y *deep learning* para realizar recomendaciones.
- Realizar una aplicación utilizable por un operario para validar diferentes bobinas.

### B.3. Catalogo de requisitos

A continuación se indican los requisitos marcados en la aplicación llevada a cabo en el proyecto:

- **REQ 1:** conseguir una aplicación capaz de ejecutar un conjunto de modelos sobre los datos de una bobina.
  - **REQ 1.1:** cargar los datos medidos por los sensores.
  - **REQ 1.2:** obtener las principales características de las bobinas.
  - **REQ 1.3:** evaluar las bobinas con los modelos.
  - **REQ 1.4:** mostrar si la bobina es válida o no.
- **REQ 2:** permitir que los resultados obtenidos puedan descargarse.
  - **REQ 2.1:** descargar las características obtenidas de las bobinas.
  - **REQ 2.2:** descargar los resultados obtenidos por el modelo.
- **REQ 3:** generar un archivo que contenga el historial de las diferentes bobinas cargadas y evaluadas.
  - **REQ 3.1:** guardar en el fichero el día en que se cargó la bobina, junto con la clase predicha.

## B.4. Especificación de requisitos

### Actores

En esta aplicación tan solo se detecta un actor: el operario de la empresa. Ya que es el encargado, una vez se cuenten con los datos obtenidos por los sensores sobre la bobina, de dirigirse a la aplicación y realizar la predicción sobre los mismos.

## Diagrama de casos de uso

La Figura B.1 muestra el diagrama de casos de uso de la aplicación.

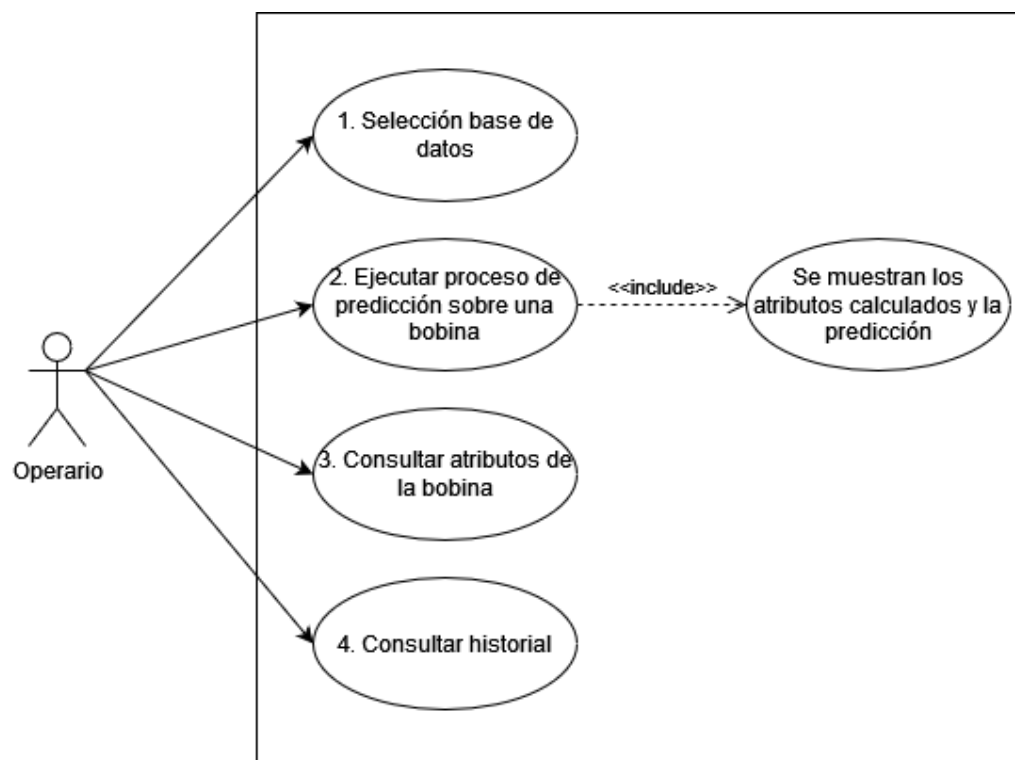


Figura B.1: Diagrama de Casos de Uso

## Especificación de casos de uso

A continuación se muestra una tabla para cada caso de uso:

<b>CU 1 - Selección base de datos</b>	
<b>Descripción</b>	Permite al operario seleccionar los diferentes valores que permitan conectarse a la base de datos para cargar los valores.
<b>Requisitos</b>	REQ 1.1
<b>Precondiciones</b>	Se está ejecutando el notebook de la aplicación.
<b>Secuencia</b>	1. El usuario rellena los campos: user, password, host y database 2. El usuario ejecuta la celda correspondiente
<b>Postcondiciones</b>	La celda se ejecuta bien y aparece un 2 a su izquierda
<b>Excepciones</b>	Se introducen los valores de forma incorrecta y sin el formato necesario en Python y salta un error

Tabla B.1: Caso de Uso - 1

<b>CU 2 - Ejecución proceso de predicción sobre una bobina</b>	
<b>Descripción</b>	Permite al operario seleccionar la ID de la bobina desea sobre la que calcular las predicciones.
<b>Requisitos</b>	REQ 1.2, REQ 1.3 y REQ 1.4
<b>Precondiciones</b>	Se está ejecutando el notebook de la aplicación, y se ha indicado la base de datos.
<b>Secuencia</b>	1. El usuario ejecuta la tercera celda. 2. El usuario selecciona en el desplegable la ID de la bobina. 3. El usuario observa los valores y predicciones mostradas por pantalla, y decide si seleccionar otra bobina.
<b>Postcondiciones</b>	Se muestran los atributos calculados de la bobina y la decisión del modelo.
<b>Excepciones</b>	Se produce algún problema durante el proceso y el modelo devuelve algún error.

Tabla B.2: Caso de Uso - 2

<b>CU 3 - Consultar atributos de la bobina</b>	
<b>Descripción</b>	Permite al operario conocer los atributos y el mapa codificado de la bobina previamente calculados.
<b>Requisitos</b>	REQ 2.1
<b>Precondiciones</b>	Se ha realizado una predicción sobre la bobina deseada.
<b>Secuencia</b>	<ol style="list-style-type: none"> <li>1. El usuario se dirige a la ruta donde se encuentra la aplicación.</li> <li>2. El usuario se dirige al directorio /bobinas.</li> <li>3. El usuario abre el CSV correspondiente a la bobina deseada.</li> </ol>
<b>Postcondiciones</b>	El usuario consigue abrir el fichero correctamente, y en su interior se encuentran todos los atributos.
<b>Excepciones</b>	-

Tabla B.3: Caso de Uso - 3

<b>CU 4 - Consultar el historial</b>	
<b>Descripción</b>	Permite al operario poder revisar el historial de las diferentes predicciones realizadas.
<b>Requisitos</b>	REQ 2.2 y REQ 3
<b>Precondiciones</b>	Se ha realizado al menos una predicción sobre cualquier bobina.
<b>Secuencia</b>	<ol style="list-style-type: none"> <li>1. El usuario se dirige a la ruta donde se encuentra la aplicación.</li> <li>2. El usuario abre el archivo historial.txt</li> </ol>
<b>Postcondiciones</b>	El archivo contiene en su interior los resultados previamente calculados.
<b>Excepciones</b>	-

Tabla B.4: Caso de Uso - 4





## *Apéndice C*

---

# Especificación de diseño

---

### C.1. Introducción

Este tercer apéndice tratará el diseño de los datos usados en la aplicación, junto con el diseño procedimental y el diseño arquitectónico.

### C.2. Diseño de datos

En este subpunto se recogen la información relevante de las tres tablas de datos que nos ha dado la empresa para poder realizar el proyecto. Además, se explican los diferentes sensores que hay, junto con su identificación para su posible comprensión. Esta descripción es muy importante, porque son los datos con los que se ha podido llevar a cabo el proyecto, y es importante que cualquier persona los pueda comprender para entender claramente que se ha efectuado en el código desarrollado.

A continuación se recoge el nombre de cada tabla junto con los atributos más relevantes (aquellos que han sido utilizados a lo largo del proyecto):

- **Gauges:** contiene un total de 6 valores, que se corresponden con los diferentes sensores de la empresa. A continuación, se muestra la ID de cada sensor junto con sus principales características:
  - **123:** se corresponde con un sensor de datos 1D que mide la capa superior de la bobina. Toma datos cada 25 metros.
  - **124:** se corresponde con la pareja del sensor 123, pero toma datos de la capa inferior de la bobina.

- **201:** se corresponde con otro sensor de datos 1D, pero diferente al anteriormente mencionado, de la capa superior. Este sensor también toma datos cada 25 metros.
  - **202:** es la pareja del sensor 201. Toma medidas de la capa inferior de la bobina.
  - **1234:** es un sensor de los datos 2D y toma datos de la capa superior de la bobina. Los datos son tomados cada 10 metros.
  - **1243:** es la pareja del sensor 1234, y toma datos de la capa inferior de la bobina.
- **GR\_LPValues:** contiene un total de 115972 registros de los datos 1D. Cada registro se corresponde para una bobina, sensor y *teja* específica. Los principales atributos son:
- **COILID:** se corresponde con la ID de la bobina.
  - **TILEID:** se corresponde con la ID de la *teja* de la bobina. Cada *teja* tiene diferente ID, su orden es cronológico, es decir, la primera *teja* será el valor más pequeño, la segunda *teja*, el segundo valor más pequeño, y así sucesivamente.
  - **MID:** se corresponde con la ID del sensor que ha capturado los datos. Estos valores son los mencionados anteriormente, dónde se explicaba como funcionaba cada sensor.
  - **MEAN:** contiene el valor medio de zinc en dicha *teja*. Este es el valor utilizado para ver si cumple o no con los requisitos de zinc.
- **GR\_QPValues:** contiene los registros de los datos 2D. En total hay 1272312 registros, y de igual manera que en los datos 1D, cada registro se corresponde para una bobina, sensor y *teja*. Los atributos usados son los mismos que los de la tabla anterior, pero con la diferencia que ahora el atributo TILEID contiene a la ID de cada celda, en vez de cada *teja*.
- **V\_Coils:** contiene las bobinas (cuyas IDs se encuentran en el atributo SID) con la etiqueta colocada por parte del operario (atributo *CLASSLABEL*). Además, tiene los valores mínimos y máximos de zinc (atributos *ZnMin* y *ZnMax*) que tiene que tener cada bobina.

## C.3. Diseño procedimental

La Figura C.1 muestra el diagrama de secuencia en el cual se puede ver como cambia el flujo a la hora de realizar una predicción sobre la bobina deseada por parte del usuario.

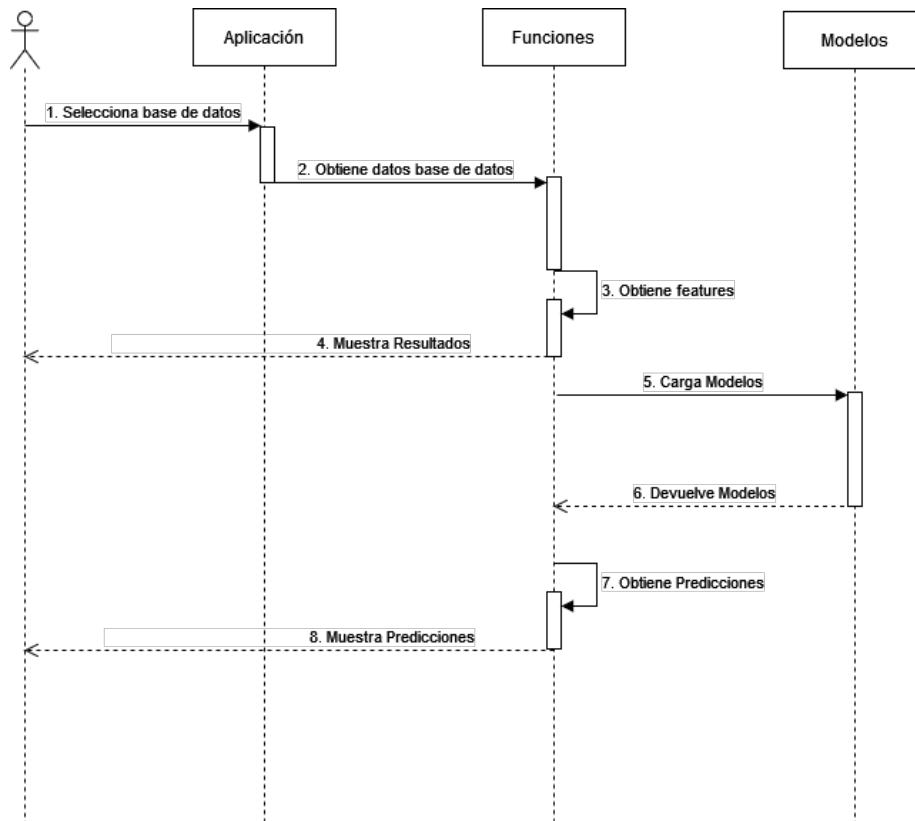


Figura C.1: Diagrama de Secuencia

## C.4. Diseño arquitectónico

En el diseño arquitectónico que compone la aplicación se pueden identificar tres elementos principales:

- **Aplicación:** es un *notebook* formado por tan solo tres celdas que permite al usuario poder seleccionar la base de datos a la que conectarse y posteriormente elegir la bobina o bobinas sobre las que se desea realizar predicciones.

- **Funciones:** es un fichero de *Python* que contiene las diversas funciones que permiten el intercambio de flujo entre la aplicación y las diferentes funciones. Además, es capaz de calcular las diferentes características y mostrar las predicciones de las bobinas. Finalmente, permite también generar CSV sobre las bobinas y llevar un registro sobre las diferentes predicciones en un fichero de texto.
- **Modelos:** son los encargados de efectuar las predicciones sobre los datos que genera la parte de funciones.

## Apéndice *D*

---

# Documentación técnica de programación

---

### D.1. Introducción

Este cuarto apéndice va destinado a las personas dedicadas a la informática y que quieran seguir con el proyecto en el futuro. Es por ello, que es necesario explicar todo claramente para que se pueda comprender. Los temas sobre los que se va a tratar son: estructura de directorios, manual del programador, ejecución del proyecto y pruebas del sistema.

### D.2. Estructura de directorios

Este subapartado contiene la información sobre la estructura del proyecto, junto con sus diferentes directorios y ficheros. Además, toda la estructura se puede ver de manera pública en el repositorio de GitHub<sup>1</sup> del proyecto. A continuación, se puede ver de manera gráfica:

```
/
├── doc
├── src
├── LICENSE
├── README.md
└── requirements.txt
```

Como se aprecia, se distinguen del directorio raíz 5 elementos, que se van a explicar a continuación:

---

<sup>1</sup>Repositorio: <https://github.com/ifh1001/TFM>

- **doc:** contiene toda la información relacionada con la documentación.
- **src:** contiene en su interior todo el código desarrollado a lo largo del proyecto. Además, contiene la aplicación, junto con todo lo necesario para que pueda usarse.
- **LICENSE:** fichero que contiene la licencia del proyecto, que como ya se ha comentado en el primer apéndice, es una licencia de tipo GPL v3.
- **README.md:** contiene una breve descripción del repositorio, con el fin de que el lector comprenda la finalidad del proyecto.
- **requirements.txt:** fichero que reúne los requisitos necesarios de librerías y versiones para ejecutar sin ningún problema la aplicación.

## Documentación

Como se ha comentado previamente, la documentación se encuentra en el directorio `/doc`. En él se encuentra un conjunto de ficheros escritos en  $\text{\LaTeX}$  que si se compilan todos ellos juntos se obtiene la memoria en formato PDF que da lugar a esta memoria que se está leyendo.

## Código

El código desarrollado en el proyecto se encuentra en el directorio `/src`. A continuación se muestra la división de directorios y ficheros del mismo:

```
/src/
├── aplicacion
│   ├── modelos
│   ├── Aplicacion.ipynb
│   ├── funciones.py
│   ├── CargaDatos.ipynb
│   ├── EvaluacionCNNs.ipynb
│   ├── ObtencionCaracteristicas.ipynb
│   ├── PreparacionCNNs.ipynb
│   └── VisualizacionDatos.ipynb
```

Como se puede apreciar, el directorio `/src` está formado por un directorio y cinco *notebooks*. A continuación, se puede ver una explicación sobre cada uno:

- **aplicacion:** en este directorio se encuentra la estructura necesaria para que funcione la aplicación. Como se puede ver, hay un directorio más, `/modelos`, donde se encuentran almacenados los cinco modelos que utilizará la aplicación. Además, hay dos ficheros, un *notebook* que será la aplicación y un fichero de *Python* que contiene todas las funciones de manera encapsulada para que se pueda utilizar la aplicación de la manera más sencilla posible.
- **CargaDatos.ipynb:** este *notebook* contiene las primeras pruebas que se realizaron sobre los datos para conocer las diferentes tablas y registros que tenía cada una. Además, contiene una evaluación de bobinas según si cumplen o no todas sus tejas, los requisitos de zinc.
- **EvaluacionCNNs.ipynb:** este fichero contiene los diferentes experimentos que se han llevado a cabo para obtener el mejor conjunto de modelos con la mayor precisión. Además, también guarda todos los modelos obtenidos para poder emplear cualquiera.
- **ObtencionCaracteristicas.ipynb:** este *notebook* se encarga de obtener el mapa codificado de cada bobina junto con las diferentes características. Todos estos datos se calculan para cada tipo de datos, 1D y 2D, y para cada uno de los diferentes sensores.
- **PreparacionCNNs.ipynb:** este fichero contiene las diferentes pruebas que se han efectuado para aprender a construir modelos con *tensorflow* para familiarizarse con la librería y sentirse cómodo con el uso de la misma.
- **VisualizacionDatos.ipynb:** contiene una visualización de los valores de zinc para cada bobina en forma de gráfica, para cada uno de los sensores de los datos 1D.

## D.3. Manual del programador

Tal y como se acaba de ver en el subpunto anterior. La aplicación se encuentra en el directorio `/src/aplicacion`. En él se pueden implementar nuevas funcionalidades de la aplicación añadiéndolas al fichero `funciones.py`. Además, en su interior, se encuentran todas las funciones debidamente comentadas para que puedan ser comprendidas y mejoradas en caso de que se desee. Adicionalmente, si se consiguen mejores modelos se pueden eliminar los antiguos y añadir los nuevos en el directorio `/src/aplicacion/modelos`.

Si se quiere realizar alguna prueba más sobre las redes neuronales, o construir algunas nuevas empleando diferentes estrategias, o incluso utilizando datos diferentes, mi recomendación es que se realice en un *notebook*, usando por ejemplo *Jupyter Notebook*, ya que permite visualizar todo de una manera mucho más sencilla y posteriormente exportar las funciones a un fichero de *Python*. Todos estos nuevos ficheros se pueden añadir directamente al directorio raíz del código (`/src`).

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

Para usar la aplicación desarrollada en el proyecto, es necesario tener instalado en el dispositivo un entorno que tenga todas las dependencias de librerías que usa la aplicación, junto con una herramienta que permita ejecutar *notebooks*, siendo recomendad el uso de *Jupyter Notebook*.

Las librerías utilizadas se pueden ver a continuación:

- MySQL
- Pandas
- NumPY
- Jupyter Widgets
- TensorFlow

De forma que al instalar todas ellas en un entorno, se podría ejecutar sin ningún problema la aplicación. Pese a todo, y por si en algún punto existen versiones incompatibles entre sí, en el repositorio de GitHub existe el fichero **requirements.txt** para que se puedan instalar en un entorno de Anaconda las versiones utilizadas en el proyecto que garantizarían el correcto empleo de la aplicación. Dicho fichero se puede encontrar en el siguiente enlace:

<https://github.com/ifh1001/TFM/blob/main/requirements.txt>

Una vez se cuenten con las librerías instaladas, se puede usar la aplicación. Para ello, con una herramienta que permita ejecutar *notebooks* se puede abrir el correspondiente a la aplicación y ejecutar las tres celdas que la componen para poder seleccionar las bobinas sobre las que realizar predicciones.



## **D.5. Pruebas del sistema**

Este proyecto se ha centrado en la investigación de obtener un grupo de modelos capaces de predecir si una bobina iba a ser válida o no para poder usarse por parte de la empresa. Es por ello, que se ha dado más peso a la investigación que a la construcción de una aplicación lo más perfecta posible y realizando pruebas de sistema para garantizar su correcto funcionamiento.

Pese a todo, se han llevado a cabo dos pruebas relevantes a la hora de construir y evaluar los modelos. Por un lado, se han separado del conjunto de datos inicial, una muestra de bobinas para, tras construir los modelos, evaluarlos con datos que nunca habían visto y así obtener unos resultados lo más realistas y probables posibles.

Por otro lado, se llevó a cabo una prueba para obtener el mejor criterio de clasificación, probando un total de 5 valores, y para cada uno de ellos se calculó y representó la matriz de confusión junto con su tasa de acierto, con el fin de identificar el mejor criterio de los 5.



## Apéndice *E*

---

# Documentación de usuario

---

### E.1. Introducción

En este último apéndice, se incluye la información necesaria para poder utilizar la aplicación del proyecto, como poder instalarlo y el manual de usuario para saber utilizar la aplicación.

### E.2. Requisitos de usuarios

El primer requisito por parte del usuario es la necesidad de tener conexión a Internet, ya que los datos se encuentran alojados en una base de datos, por lo que para poder usarla será necesario tener conexión.

En segundo lugar, necesitará tener un entorno con todas las librerías que usa la aplicación, junto con alguna aplicación que permite ejecutar un *notebook*, como, por ejemplo, *Jupyter Notebook*.

Una vez se diponga de todo lo anterior, ya sería posible poder utilizar la aplicación que se ha llevado a cabo en este proyecto.

### E.3. Instalación

En primer lugar, sería necesario descargar e instalar Anaconda en caso de que no se disponga de dicha herramienta, ya que permite crear un entorno con todos los requisitos de la aplicación, permitiendo que se pueda usar sin ningún problema. Para ello, es tan sencillo como ir a la siguiente ruta:

<https://www.anaconda.com/download>

Y en ella descargar la versión adecuada a tu sistema operativo. Tras esto, sería tan sencillo como seguir el instalador y esperar unos minutos a que termine la instalación.

En segundo lugar, habría que instalar las dependencias de la aplicación. Para facilitar este proceso se ha dejado en el repositorio del proyecto un fichero *requirements*, obtenido de un entorno *Windows*, que permitirá descargar todas las librerías junto con sus versiones que hacen que se garantice la posibilidad de utilizar la aplicación. Dicho fichero se puede encontrar en el siguiente enlace:

<https://github.com/ifh1001/TFM/blob/main/requirements.txt>

En tercer, y último lugar, para poder instalar las dependencias, se abrirá la consola *Anaconda Prompt* y se escribirá el siguiente comando:

```
conda create --name nombre_entorno --file requirements.txt
```

Y tras esperar unos minutos, se habrán instalado todas las dependencias, por lo que ya se ha creado un entorno que sea capaz de lanzar la aplicación.

## E.4. Manual del usuario

En primer paso para poder utilizar la aplicación, es lanzar *Jupyter Notebook*. Para ello, y basándose en el apartado de instalación, lo primero sería cambiar al nuevo entorno, y posteriormente ejecutar el comando de la herramienta. Ambos comandos se muestran a continuación:

```
conda activate nombre_entorno
```

```
jupyter notebook
```

Tras su ejecución aparecerá en el navegador la herramienta, y sería tan sencillo como dirigirse al directorio donde se encuentra el *notebook* de la aplicación. Tras abrirlo se verá que hay 3 celdas y que habrá que ejecutar en orden. Para ejecutar una celda, hacemos clic sobre ella y pulsamos al botón **Run** que se encuentra en el menú superior, tras esperar un rato veremos como a la izquierda de la celda aparecerá un 1, de igual forma que se puede ver en la Figura E.1.

---

```
In [1]: from funciones import cargaDatos
```

---

Figura E.1: Ejecución de la Primera Celda

A continuación, será necesario ejecutar la segunda celda, o previamente modificar los valores en caso de que se quiera utilizar una base de datos diferente a la que viene por defecto. Tras su ejecución aparecerá a la izquierda un 2.

Finalmente, se ejecutará la última celda, y se verá como aparece un desplegable que muestra las IDs de todas las bobinas cargadas de la base de datos, Figura E.2. Sería tan sencillo como seleccionar una y esperar a que aparezcan los resultados. Este proceso se puede repetir tantas veces como se desee, simplemente volviendo a seleccionar otra bobina desde el desplegable.

### Predicción de bobinas

---

```
In [3]: cargaDatos(user, password, host, database)
```

---

Selecciona...  ▼

Figura E.2: Ejecución de la Tercera Celda

También, en el directorio `./bobinas` se verán los diferentes CSV asociados a las bobinas sobre las que se han realizado predicciones. Además, junto al *notebook* de la aplicación, se encontrará el fichero `historial.txt` donde se podrá consultar las predicciones de todas las bobinas.

Tras usar la aplicación, sería tan sencillo como cerrar la pestaña de la aplicación y finalmente cerrar la consola desde la que se ha lanzado *Jupyter Notebook*, y de esta forma se detendría todo el proceso por completo.



---

## Bibliografía

---

- [1] <https://proyectosagiles.org/que-es-scrum/>, journal=Proyectos Ágiles, Sep 2021.
- [2] Jesús Bobadilla. *Machine learning y deep learning: usando Python, Scikit y Keras*. Ediciones de la U, 2021.
- [3] Wikimedia Commons. File:convolution arithmetic - padding strides odd.gif — wikimedia commons, the free media repository, 2020. [Online; fecha de acceso 27-Junio-2023].
- [4] Wikimedia Commons. File:max pooling.png — wikimedia commons, the free media repository, 2021. [Online; fecha de acceso 27-Junio-2023].
- [5] Ferros Planes. Proceso de galvanizado - ventajas y aplicaciones. <https://ferrosplanes.com/proceso-galvanizado-ventajas/>, 2018.
- [6] A González-Marcos, JB Ordieres-Meré, AV Pernía-Espinoza, and V Torre-Suárez. Desarrollo de un cerrojo artificial para el skin-pass en una línea de acero galvanizado por inmersión en caliente. *Revista de metalurgia*, 44(1):29–38, 2008.
- [7] Pablo Haya. La metodología crisp-dm en ciencia de datos - iic. <https://www.iic.uam.es/innovacion/metodologia-crisp-dm-ciencia-de-datos/>, Jan 2022.
- [8] Fernando Izaurieta and Carlos Saavedra. Redes neuronales artificiales. *Departamento de Física, Universidad de Concepción Chile*, 2000.
- [9] Zhao Lu, Yimin Liu, and Shi Zhong. Research on zinc layer thickness prediction based on lstm neural network. In *2021 33rd Chinese Control and Decision Conference (CCDC)*, pages 4995–4999, 2021.

- [10] Kai Mao, Yong-Li Yang, Zhe Huang, and Dan-yang Yang. Coating thickness modeling and prediction for hot-dip galvanized steel strip based on ga-bp neural network. In *2020 Chinese Control And Decision Conference (CCDC)*, pages 3484–3489, 2020.
- [11] Na8. Clasificación con datos desbalanceados: principales soluciones. <https://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/>, 2020.
- [12] Juan Sebastian Gelvez Prieto. Redes neuronales convolucionales y redes neuronales recurrentes en la transcripción automática, 2019.