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 — 8 de julio de 2023
Tutores: Carlos Enrique Vivaracho Pascual
Joaquín B. Ordieres Meré

Universidades de Burgos, León y Valladolid

Máster universitario en Inteligencia de Negocio y Big Data en Entornos Seguros

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Ismael Franco Hernando, con DNI 71363577M, ha realizado el Trabajo final de Máster en Inteligencia de Negocio y Big Data en Entornos Seguros titulado Aplicación de Machine Learning a imágenes 1D y 2D de control de calidad en recubrimiento de Zinc.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 8 de julio de 2023

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

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, para ver cuál de ellos da mejores resultados junto con la validación cruzada, 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 de 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 zinc layer levels within certain minimums to ensure that the steel withstands different scenarios it is intended for, as well as maximums to prevent the company from losing money by overspending on zinc and avoid excessive hardening of the steel, making it difficult 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 defect, can make a prediction on whether the coil will be usable or not.

To achieve this, various experiments will be conducted on 1D and 2D data to determine which of them yields better results, along with cross-validation, while testing different parameters to obtain the best possible outcome.

Additionally, an application will be developed to simulate the process that a company operator should follow to load coil data and wait for the predictions on them.

Keywords

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

Índice general

Índice de figuras

Índice de tablas

Memoria

Introducción

El galvanizado [4] 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 interior, ya que otorga una mejor estética.

Cabe destacar que esta técnica se usa principalmente 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 como 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 que es el galvanizado, como 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 obtener 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 [5] es difícil, ya que siempre el acero 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 aquella 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 memoria está formada por los siguientes puntos:

1. **Introducción:** pone en contexto al lector sobre el problema al que se enfrenta el proyecto. Además, contiene la estructura de toda la documentación.

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:

- Ayudar a la empresa que origina el proyecto, consiguiendo que se puedan identificar el mayor número de bobinas con errores aprovechables por parte de la empresa.
- Aumentar la eficiencia de la cadena de galvanizado de la empresa, de manera que un operario no tenga que analizar de forma exhaustiva cada bobina, para saber si es utilizable o no.
- Crear una cadena más sostenible, consiguiendo que se puedan aprovechar al máximo las bobinas galvanizadas y evitar que se tenga que desperdiciar demasiado material.

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 prestado 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:

- Conocer el proceso de galvanizado, con el fin de poder interpretar y desarrollar de la mejora manera posible el proyecto.
- Aplicar los conocimientos adquiridos a lo largo del máster universitario.
- Profundizar, mejorar y aplicar los conocimientos sobre el lenguaje de programación *Python*.

Trabajos relacionados

Analizando trabajos que también usen el *deep learning* en la rama de la industria centrada en el acero galvanizado, me he dado cuenta de que existen 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 detectar defectos en el galvanizado de láminas de acero con redes neuronales. A continuación se muestran algunos de ellos:

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

Autores: Zhao Lu, Yimin Liu y Shi Zhong

Este artículo [7] utiliza una red neuronal LSTM, 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 [8] es muy parecido al anterior, solo que utilizan una red neuronal BP, 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 genérico 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, 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, se basa en 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 e infinitos escenarios y posibilidades, y que a su vez se puedan normalizar todas sus soluciones de forma matemática.

Deep Learning

El aprendizaje profundo, también conocido como *deep learning* en inglés, 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.

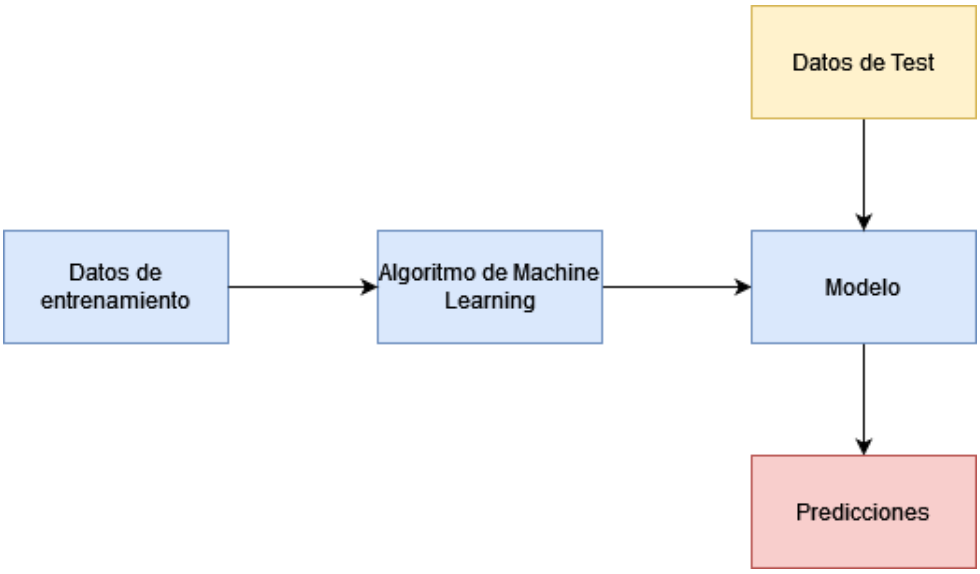

Figura 4.1: Esquema Algoritmos Machine Learning (Realización Propia).

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 únicamente aprenden de manera iterativa, mientras que esta rama es capaz de aprender mucho más rápido debido a su profundidad.

4.2. Redes Neuronales Artificiales

Las redes neuronales artificiales [6] buscan recrear el sistema nervioso del ser humano, donde existen multitud de neuronas conectadas entre sí, y que intercambian pulsos.

Es por ello que 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.

Estas redes están tan extendidas debido a que no hay que programar neurona a neurona, para asignarles una tarea en específico, ya que la propia red aprende sola. Esto es así, debido a que busca minimizar una función de pérdida de la red al completo, lo que permite que se pueda encontrar dentro de la red el camino más óptimo.

Según como fluya el flujo de 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*.

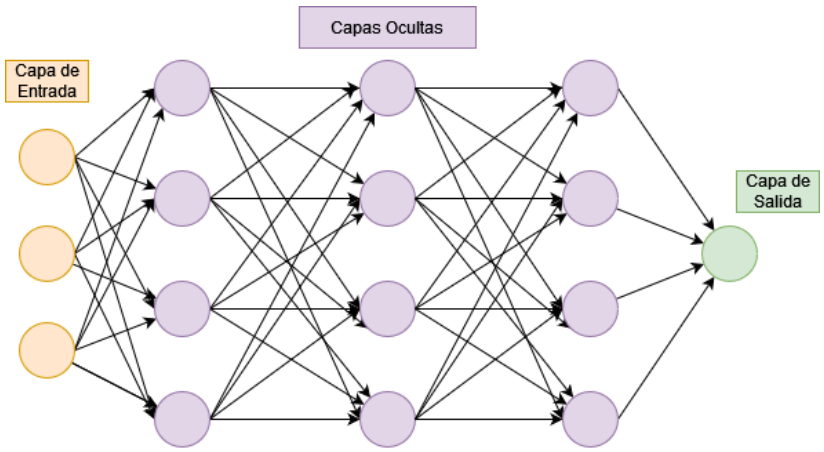

Figura 4.2: Ejemplo Red Neuronal Artificial (Realización Propia)

Redes Neuronales Convolucionales

Las redes neuronales convolucionales [11], 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, pesé 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 es una operación que busca obtener características de los datos para que puedan ser aprendidos por la red neuronal. Para ello se ha utilizado 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 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.

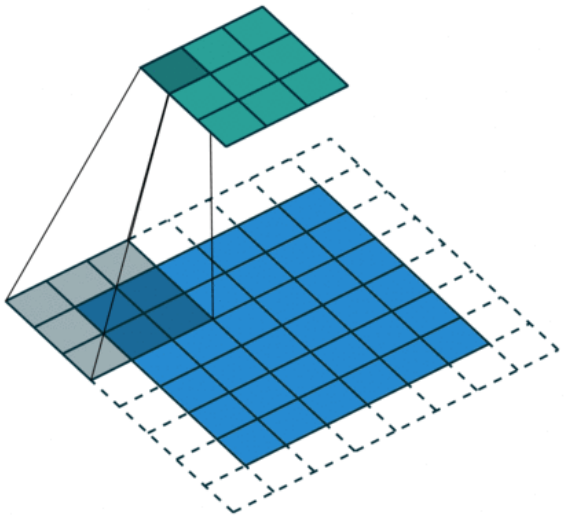

Figura 4.3: Operación de Convolución [1]

Pooling

La operación *pooling* 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 cuadrícula de un tamaño en específico y para cada celda se devuelve un valor, que generalmente 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.

La Figura 4.4 muestra un ejemplo de *pooling* aplicando una cuadrícula de 2x2 y devolviendo por cada celda el valor máximo.

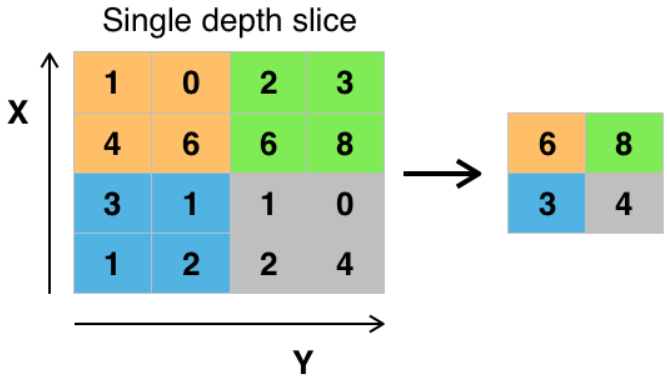

Figura 4.4: Operación de *Pooling* [2]

4.3. Conjunto de Datos Desbalanceado

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 este sana a que la gente esté enferma, por lo que en este conjunto de datos sería difícil de tener número de personas equilibrado de ambas clases. 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 [10]:

- **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 «sintéticos» para poder equilibrar los datos de cada clase.

- **Sobremuestreo y submuestreo**: suele ser la estrategia más frecuente, donde 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.

4.4. Curva ROC (Receiver Operating Characteristic)

La curva ROC [9] es una representación gráfica utilizada principalmente en el aprendizaje automático para clasificación binaria y en estadística. Dicha representación muestra como varían los verdaderos y falsos positivos según cambia el criterio discriminatorio entre las clases. De modo que se pueda cambiar el criterio y ver a su vez como varían las diferentes tasas de ciertos y falsos positivos.

Además, la curva ROC va acompañada del valor AUC (*Area Under Curve*), valor comprendido entre 0 y 1, de forma que cuanto mayor sea el área bajo la curva, el criterio discriminante entre las clases será también mejor. De forma que para comparar los diferentes criterios discriminantes se empleará el valor AUC, para elegir aquel con un valor mayor.

En la Figura 4.5 se puede ver un ejemplo de tres curvas ROC (tres clasificadores diferentes), donde la mejor es la de color azul. Además, muestra en el punto situado en la esquina superior izquierda por donde debería de pasar la curva ideal para obtener un AUC igual a 1, es decir, un clasificador que identifica correctamente las dos clases.

Según el AUC obtenido, el modelo será mejor o peor, y es por esta razón que se establece a modo de guía la siguiente interpretación del valor AUC [12]:

- **0 - 0.5:** clasificador demasiado malo, ya que sería peor o igual que tener un clasificador aleatorio con probabilidad 0.5 para cada clase.
- **0.5 - 0.6:** clasificador malo.
- **0.6 - 0.75:** clasificador regular.
- **0.75 - 0.9:** clasificador bueno.
- **0.9 - 0.97:** clasificador muy bueno.
- **0.97 - 1:** clasificador excelente.

En definitiva, lo mejor sería obtener un clasificador con AUC mayor o igual a 0.75, ya que los resultados que se obtendrán serán seguramente muy buenos.

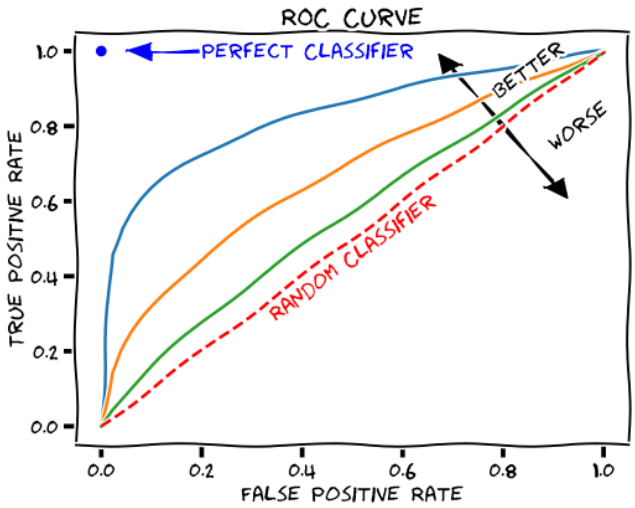

Figura 4.5: Ejemplo de curva ROC para tres clasificadores [3]

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 ágil del *Scrum*. En dicha metodología se pretende realizar las diferentes tareas de manera incremental, formando un *sprint*. Estos últimos tenían una duración de dos semanas, donde al pasar el tiempo se llevaba a cabo una reunión para revisar el progreso y tomar las decisiones adecuadas en el correcto desarrollo del proyecto.

Con respecto al entendimiento y procesado de los datos se ha seguido la metodología CRISP-DM, es decir, se han realizado diferentes etapas (entendimiento, preparación, modelado, evaluación y despliegue) con el conjunto de datos original.

5.2. Gestión del Proyecto

GitHub

GitHub[1] 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.

[1]GitHub: https://github.com/

5.3. Herramientas

Anaconda

Anaconda[2] 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.

Jupyter Notebook

Jupyter Notebook[3] es una interfaz web utilizada principalmente para ejecutar sentencias de código desde el navegador a través de los *notebooks* que permite generar.

5.4. Bibliotecas de Python

Durante todo el desarollo 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*[4] permite manipular y analizar grandes cantidades de datos. Se asemeja mucho a la herramienta de *Excel* pero en *Python*.

NumPy

NumPy[5] permite utilizar arrays y matrices sobre las cuales realizar las principales operaciones matemáticas. Además, son muy útilies frente a grandes conjuntos de datos, ya que las operaciones están muy optimizadas.

MySQL

La librería *MySQL*[6] 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.

[2] Anaconda: https://www.anaconda.com/
[3] Jupyter Notebook: https://jupyter.org/
[4] Pandas: https://pandas.pydata.org/
[5] NumPy: https://numpy.org/
[6] 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[7] 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*[8] lo que simplifica y facilita la construcción de redes neuronales.

Scikit-Learn

La librería *Scikit-learn*[9] 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[10] se centra en abordar el problema de clases desbalanceadas, de modo que tiene múltiples herramientas para intentar de solventar este inconveniente.

Matplotlib

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

---

[7] TensorFlow:https://www.tensorflow.org
[8] Keras:https://keras.io/
[9] Scikit-learn:https://scikit-learn.org/
[10] Imbalanced-learn:https://imbalanced-learn.org/
[11] Matplotlib: https://matplotlib.org/

Jupyter Widgets

Jupyter Widgets[12] 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.

5.5. Documentación

LaTeX

LaTeX[13] es un sistema de composición de textos usado principalmente en la escritura de textos con alta calidad tipográfica.

Overleaf

Overleaf[14] es un editor colaborativo de LaTeX cuyo principal uso es escribir, editar y publicar documentos científicos, todo ello gestionado desde la nube.

Draw.io

Draw.io[15] 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[16] es una web que facilita la creación de tablas para LaTeX, ya que se pueden diseñar a través de una interfaz y posteriormente exportarlo a código.

[12] Jupyter Widgets: https://github.com/jupyter-widgets/ipywidgets
[13] LaTeX: https://www.latex-project.org/
[14] Overleaf: https://www.overleaf.com/
[15] Draw.io: https://app.diagrams.net/
[16] Tables Generator: https://www.tablesgenerator.com/#

Aspectos relevantes del desarrollo del proyecto

Este punto es el más importante de la memoria, ya que contiene, de manera descriptiva y detallada, el proceso seguido desde el inicio hasta el fin del proyecto. Para una mejor comprensión, el orden de los subpuntos que contiene este apartado están ordenados de manera cronológica. Además, se hablará acerca de los problemas surgidos, soluciones tomadas y alguna posible alterativa.

Antes de entrar en materia, cabe destacar que la primera idea que teníamos por parte de la empresa, es que querían comprobar, si con los niveles mínimos y máximos de zinc, la decisión por parte del operario era la correcta acerca de si la bobina era válida o no. Y tras comprobar los niveles y clasificar de nuevo las bobinas, crear una red neuronal capaz de clasificar futuras bobinas.

Pero más adelante se nos comentó que las decisiones por parte del operario sobre si la bobina era válida o no, eran las correctas, y su interés estaba en obtener un modelo que con los valores capturados por los sensores se pueda predecir, sin necesidad del operario, si la bobina es correcta o no.

6.1. Carga de Datos y Evaluación de Bobinas

En esta primera fase, lo primero que se hizo fue cargar los datos que nos había cedido la empresa, y que se encontraban en una base de datos

de MySQL, para realizar una sencilla exploración de datos y saber con exactitud el número de datos con el que se contaba.

Antes de nada, comentar que la empresa nos cedió datos de bobinas medidas por sensores 1D y 2D, por lo que se cuenta con dos tipos de datos. Además, de los sensores 1D se contaba con dos parejas de sensores diferentes (cada pareja tenía un sensor que media la capa de arriba y otro la capa de abajo de la bobina.) De manera gráfica se entiende muy fácilmente cómo son los datos con los que se cuenta. En la Figura 6.6se puede ver como son los datos 1D, que no dejarían de ser más que un *array* de una dimensión. Con respecto a los datos 2D, Figura 6.7, se pueden interpretar como una matriz de 9 filas y el número de columnas cambiantes según cada bobina. Además, a cada celda, tanto en datos 1D como 2D, se le conoce como teja.

1	2	3	4	5	6	7

Figura 6.6: Ejemplo estructura datos 1D

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 6.7: Ejemplo estructura datos 2D

Una vez explicada la estructura de los diferentes datos, añadir que se cuentan 1160 bobinas previamente evaluadas por un operario, con 115972

registros de los datos 1D, correspondientes a las diferentes tejas medidas, y con 127312 registros de los datos 2D.

Ahora que ya se comprendían los datos, era turno de evaluar cada bobina. Para ello se creó una función que evaluaba cada bobina con sus diferentes tejas y se comprobaba que en cada una de ellas se cumplían con los requisitos mínimos y máximos de zinc correspondientes a la bobina. Toda esta evaluación se realizaba para cada uno de los sensores, y además, se contaban el número de fallos, es decir, el número de tejas que no cumplía los requisitos, y si se había producido algún calibrado. Esto último consiste en reiniciar los sensores y sus mediciones serían 0 en ese registro, pero que no indica que la teja este mal.

En la Figura 6.8 se muestra el resultado obtenido para las dos primeras bobinas, en las que se indica la ID de la bobina (*COILID*), la ID del sensor (*MID*), la clase del operario y la clase obtenida por el proyecto, y finalmente el número de fallos y el número de calibrados, si es que se ha producido alguno.

	COILID	MID	CLASSLABEL	VERIFIEDLABEL	FAILURES	CALIBRATED
0	225216688	123.0	OK	NOK	6	0
1	225216688	124.0	OK	NOK	4	0
2	225216688	201.0	OK	NOK	2	0
3	225216688	202.0	OK	NOK	2	0
4	225220725	123.0	OK	NOK	7	0
5	225220725	124.0	OK	NOK	6	0
6	225220725	201.0	OK	NOK	1	0
7	225220725	202.0	OK	NOK	3	0

Figura 6.8: Ejemplo de Evaluación de 2 Bobinas

Finalmente, en esta fase se mostraron las estadísticas de las clases obtenidas por los operarios y por el proyecto. A continuación, se puede ver en detalle:

```
Los valores de los operarios han sido:
 OK    3340
NOK    1300
```

```
Los valores que se han obtenido son:
 OK      98
NOK    4542
```

A simple vista se puede observar que había una anomalía, y como ya se había comentado previamente, se debía a esa pequeña confusión que hubo en un primer lugar con la empresa, y que tras una nueva reunión, se dijo que estas bobinas sí que estaban ya bien clasificadas previamente por un operario y que simplemente era necesario proceder a la construcción de una red neuronal.

También se puede ver como el conjunto de datos está desbalanceado (mirando las estadísticas de las clases de los operarios), ya que la clase OK es la mayoritaria, con aproximadamente un 72 % de los datos, mientras que la clase NOK es la minoritaria con aproximadamente el 28 % de los datos.

6.2. Visualización de las Bobinas

En esta segunda fase, se realizó la parte correspondiente a representar gráficamente los datos 1D junto con los valores en los que tiene que encontrarse el zinc. Para ello, se realizó un desplegable con las IDs de las diferentes bobinas, y que al seleccionar una, se mostraran los gráficos de los cuatro sensores.

La Figura 6.9 muestra un ejemplo de visualización de las cuatro gráficas, correspondientes a los diferentes sensores, junto con los diferentes valores de zinc medidos; además de una línea roja que representa el máximo o el mínimo de zinc en cada caso.

6.3. Obtención de las Características de las Bobinas

Esta tercera fase se ha centrado en la obtención de las principales *features* de cada bobina, junto con la codificación del mapa que forman los datos 1D o 2D, para posteriormente ser utilizados en la red neuronal.

En primer lugar, las características o *features* que se han obtenido para cada bobina han sido las siguientes:

- **ZNMAX_FAILURES:** se corresponde al número de tejas que tiene una capa de zinc superior al estipulado. Los fallos se contarán desde

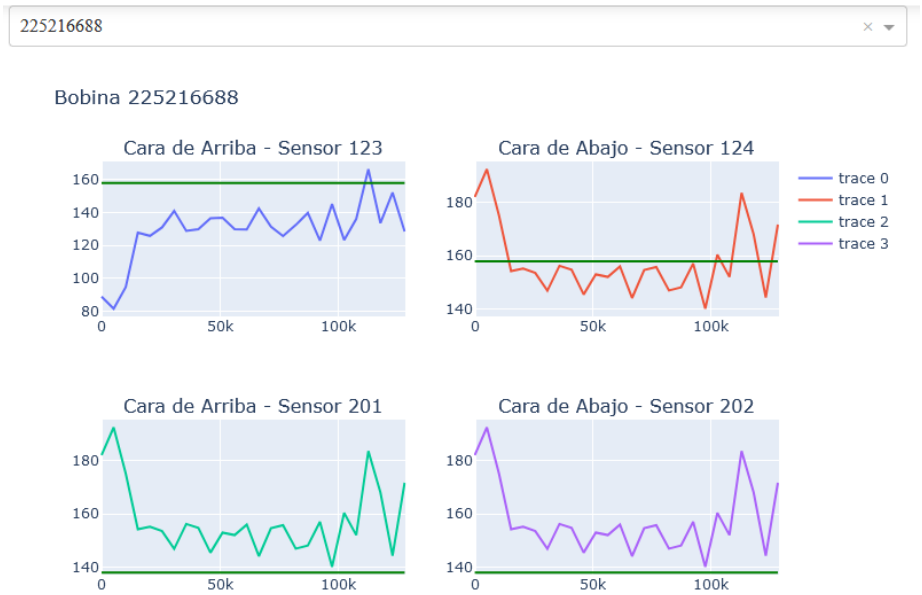

Figura 6.9: Ejemplo de Visualización para una Bobina

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.

En segundo lugar, la codificación del mapa de las bobinas se ha llevado de la siguiente manera:

- **1:** en caso de que la teja tenga más zinc del máximo.

- **0:** indica que el valor de zinc de la teja está dentro de los mínimos y máximos.
- **-1:** correspondiente a una teja que tenga menos zinc del mínimo necesario.

Cabe destacar que, tanto las *features* como la codificación del mapa de la bobina son iguales tanto para los datos 1D como 2D. Para visualizarlo mejor, la Figura 6.10, muestra un ejemplo de salida de datos 1D. En ella se pueden ver todos los atributos anteriormente mencionados para una misma bobina y los valores obtenidos por parte de los 4 sensores diferentes.

COILID	MID	ZNMAX_FAILURES	ZNMIN_FAILURES	CALIBRATED	TOTAL_TILEID	L_DIS	R_DIS	MAP	DECISSION_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, 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, 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 6.10: Ejemplo de características obtenidas para datos 1D

Por otro lado, tenemos la Figura 6.11, que muestra, para la misma bobina que en la Figura anterior, las características obtenidas para los sensores 2D. En dicha figura, se puede ver como el mapa es también un *array*, ya que por comodidad se ha guardado así, pero posteriormente, cuando se lea, se reconstruirá en forma de matriz.

COILID	MID	ZNMAX_FAILURES	ZNMIN_FAILURES	CALIBRATED	TOTAL_TILEID	L_DIS	R_DIS	MAP	DECISSION_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, 1,...	OK

Figura 6.11: Ejemplo de características obtenidas para datos 2D

Finalmente, en esta tercera fase, se han guardado las características obtenidas de los datos 1D y 2D en una base de datos. Para ello se han creado dos tablas, FEATURES_1D y FEATURES_2D, donde se almacena todo el contenido previamente calculado.

6.4. Pruebas CNNs

Una vez se contaba con los datos preparados, en este cuarta fase se realizaron diferentes pruebas en la construcción de CNNs de la librería *TensorFlow*, con el fin de poder familiarizarse con su uso. Además, y antes de empezar a construir redes neuronales, había que preprocesar los datos

para poder usarlos. También, indicar que en esta fase las pruebas fueron únicamente con datos 1D.

En primer lugar, se cargaron los mapas codificados de las bobinas 1D, obtenidos en la tercera fase, 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, con el fin de poder utilizar las convoluciones 1D sobre los datos.

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.

Ahora ye se contaba con los mapas preparados para ser usados, pero antes había que transformar la clase de *string* a *int*. Dicha transforamción fue: un 0 para la clase OK y un 1 para la clase NOK.

En este punto ya se cuenta con los datos listos, por lo que era turno de construir modelos. En primer lugar, se crearon modelos individuales para aprender como funciona su construcción, la Figura 6.12, muestra un ejemplo de construcción. Aunque, como suele ser habitual en estos casos para generar modelos, se utilizará la estrategia de validación cruzada o *cross validation* para poder evaluar los diferentes modelos construidos de manera más precisa y robusta.

Finalmente, en esta fase de pruebas también se decide 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 crear 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 esta fase, indicar que los mejores resultados se consiguieron con datos a los que se les había aplicado el sobremuestreo, por lo que se decidirá usar este conjunto de datos en la siguiente fase.

```
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 6.12: Ejemplo de Construcción de Modelo

6.5. Generación, Obtención y Evaluación CNNs

En esta quinta y penúltima fase, es turno de realizar un conjunto de experimentos, ahora que ya se sabe como crear modelos, para ver cuál es el que consigue mejores resultados para usarse en la aplicación del proyecto.

En primer lugar, se reconstruirá el mapa de los datos 2D, puesto que el de los datos 1D es el mismo que el comentado en la anterior fase. Para ello, se carga el *array* que contenía todos los datos y se pasa a una matriz de 9 filas y de nuevo se aplica la técnica de *padding*, para que todos los datos tengan el mismo número de columnas, rellenando con ceros.

Además, también se utilizarán las *features* generadas en la tercera fase con el fin de ver si pasándole al modelo el mapa y sus características normalizadas se obtienen mejores resultados. También, se probará si para estos datos funciona mejor un *Random Forest* que se construirá para cada experimento con los atributos normalizados y las predicciones de los modelos.

En segundo lugar, se ha buscado el criterio de clasificación, ya que en un primer lugar se puede pensar qué 0.5 sería el valor correcto para diferenciar ambas clases, pero la realidad es que en cada problema el criterio puede ser totalmente distinto. Para ello, se ha aplicado la técnica de validación cruzada y se ha construido un modelo y se han probado 6 criterios diferentes de clasificación: 0.2, 0.3, 0.4, 0.5, 0.6, 0.7.

Para cada uno de los criterios probados se ha construido su curva ROC y se ha calculado su AUC para ver cuál de todos ellos es el mejor. Tras realizar varias pruebas, se ha llegado a la conclusión que el mejor parámetro es 0.3, tal y como se puede ver en la Figura 6.13, donde este criterio es el que tiene mayor AUC.

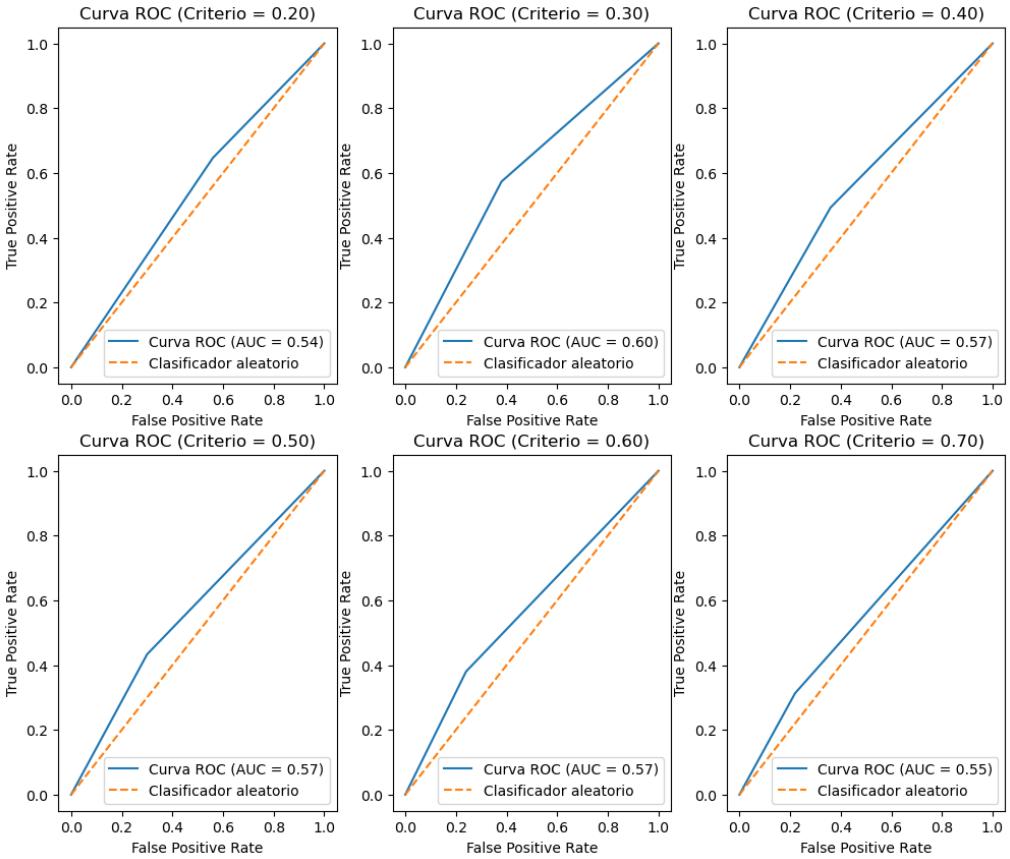

Figura 6.13: Obtención del Mejor Criterio de Clasificación

En tercer lugar, se ha separado 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 aproximada

En este momento ya se contaba con todo listo para poder realizar diferentes experimentos para construir diferentes conjuntos de modelos, cambiando los parámetros que permiten construir las diferentes redes neuronales para obtener los mejores. Para todos los experimentos se ha seguido la siguiente estrategia:

- **Capa convolucional:** esta capa o capas, según las que se le indiquen, junto con el *kernel* indicado y su activación tipo *relu*, 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*.

Además, a la hora de entrenar el modelo, se ha utilizado un *checkpoint* para quedarse con el modelo que menor *loss* tenga. También, se pueden modificar el número de iteraciones durante el entreno.

Todo esto aplicando la técnica de validación cruzada y construyendo 5 modelos en cada experimento. Por lo que a la hora de obtener predicciones se sumará la predicción de los 5 modelos y se dividirá entre 5, y finalmente este valor se comparará con el criterio de clasificación, 0.3, y se determinará la clase de la bobina en concreto.

A continuación se recogen los resultados obtenidos tanto para los datos 1D como 2D, y para cada uno de ellos, los resultados obtenidos según si se utiliza solo el mapa codificado de la bobina, o el mapa junto con los demás atributos normalizados.

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 confusion obtenida es:
 [[27 23]
 [70 80]]
```

- **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 confusion obtenida es:
 [[30 20]
 [72 78]]
```

- **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:

```
La precisión obtenida ha sido 0.54
La matriz de confusion obtenida es:
 [[29 21]
 [71 79]]
```

- **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 confusion obtenida es:
 [[29 21]
 [67 83]]
```

- **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 mimas que las del experimento anterior. Los resultados obtenidos han sido;

```
La precisión obtenida ha sido 0.57
La matriz de confusion obtenida es:
 [[32 18]
```

```
 [68 82]]
```

- **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 confusion obtenida es:
 [[30 20]
 [68 82]]
```

- **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
La matriz de confusion obtenida es:
 [[26 24]
 [75 75]]
```

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 1:** este primer experimento ha sido con una capa convolucional con 32 filtros y con un *kernel* de tamaño 3. Además, ternía 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 confusion obtenida es:
 [[35 15]
 [91 59]]

La precisión obtenida con el RF ha sido 0.475
La matriz de confusion obtenida es:
```

```
 [[38 12]
 [93 57]]
```

- **Experimento 2:** 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 confusion obtenida es:
 [[33 17]
 [81 69]]

La precisión obtenida con el RF ha sido 0.445
La matriz de confusion obtenida es:
 [[38 12]
 [99 51]]
```

- **Experimento 3:** 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 confusion obtenida es:
 [[39 11]
 [92 58]]

La precisión obtenida con el RF ha sido 0.46
La matriz de confusion obtenida es:
 [[40 10]
 [98 52]]
```

- **Experimento 4:** 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 confusion obtenida es:
 [[35 15]
 [80 70]]
```

```
La precisión obtenida con el RF ha sido 0.47
La matriz de confusion obtenida es:
 [[35 15]
 [91 59]]
```

- **Experimento 5:** 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 confusion obtenida es:
 [[34 16]
 [89 61]]

La precisión obtenida con el RF ha sido 0.455
La matriz de confusion obtenida es:
 [[40 10]
 [99 51]]
```

- **Experimento 6:** 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 confusion obtenida es:
 [[31 19]
 [85 65]]

La precisión obtenida con el RF ha sido 0.455
La matriz de confusion obtenida es:
 [[31 19]
 [90 60]]
```

- **Experimento 7:** 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 confusion obtenida es:
```

```
 [[ 39  11]
 [104  46]]

La precisión obtenida con el RF ha sido 0.435
La matriz de confusion obtenida es:
 [[ 43   7]
 [106  44]]
```

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 1:** 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 confusion obtenida es:
 [[37 13]
 [90 60]]
```

- **Experimento 2:** 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 confusion obtenida es:
 [[37 13]
 [89 61]]
```

- **Experimento 3:** 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
La matriz de confusion obtenida es:
 [[34 16]
```

```
 [79 71]]
```

- **Experimento 4:** 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 confusion obtenida es:
 [[41  9]
 [85 65]]
```

- **Experimento 5:** 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 confusion obtenida es:
 [[38 12]
 [89 61]]
```

- **Experimento 6:** 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 confusion obtenida es:
 [[36 14]
 [88 62]]
```

- **Experimento 7:** 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 confusion obtenida es:
 [[34 16]
 [76 74]]
```

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

- **Experimento 1:** 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 confusion obtenida es:
 [[41  9]
 [89 61]]

La precisión obtenida con el RF ha sido 0.5
La matriz de confusion obtenida es:
 [[42  8]
 [92 58]]
```

- **Experimento 2:** 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 confusion obtenida es:
 [[30 20]
 [79 71]]

La precisión obtenida con el RF ha sido 0.53
La matriz de confusion obtenida es:
 [[39 11]
 [83 67]]
```

- **Experimento 3:** 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 confusion obtenida es:
 [[34 16]
```

```
 [79 71]]

La precisión obtenida con el RF ha sido 0.51
La matriz de confusion obtenida es:
 [[42  8]
 [90 60]]
```

- **Experimento 4:** 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 confusion obtenida es:
 [[40 10]
 [92 58]]

La precisión obtenida con el RF ha sido 0.47
La matriz de confusion obtenida es:
 [[43  7]
 [99 51]]
```

- **Experimento 5:** 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 confusion obtenida es:
 [[38 12]
 [85 65]]

La precisión obtenida con el RF ha sido 0.495
La matriz de confusion obtenida es:
 [[37 13]
 [88 62]]
```

- **Experimento 6:** en este sexto caso, similar al primero, se han añadido dos capas convolucionaales con 32 y 16 filtros. Los resultados son:

```
La precisión obtenida ha sido 0.51
```

```
La matriz de confusion obtenida es:
 [[35 15]
 [83 67]]

La precisión obtenida con el RF ha sido 0.515
La matriz de confusion obtenida es:
 [[42  8]
 [89 61]]
```

- **Experimento 7:** 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 confusion obtenida es:
 [[39 11]
 [91 59]]

La precisión obtenida con el RF ha sido 0.475
La matriz de confusion obtenida es:
 [[42  8]
 [97 53]]
```

Análisis de resultados

Para concluir con esta fase, es turno de analizar los resultados que se han obtenido, y que han sido mostrados en los dos subpuntos anteriores.

Lo primero que resalta son los malos resultados, ya que ninguno de los 28 experimentos han mostrado una precisión adecuada, de hecho, casi todos están con una precisión de entre el 50 y 60 porciento, unos resultados para nada aceptables.

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 chocar, 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 prestados 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 el modelo, por lo que no muestra ningún síntoma de ser beneficioso su uso.

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. Desarrollo de la Aplicación

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 vas 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. Y 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 6.14, donde se aprecia la tabla con las diferentes *features* 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.

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 6.15, muestra el CSV generado para la bobina de ejemplo mostrada en la imagen anterior.

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, -1, -1, 0, ...
1	226861024	124.0	1	0	0	35	10	0	[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, 0, 0, 0, 0, 0,...
3	226861024	202.0	1	3	0	35	9	14	[0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 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 6.14: Ejemplo de Resultado de la Aplicación

226861024.csv

```
COILID,MID,ZNMAX_FAILURES,ZNMIN_FAILURES,CALIBRATED,TOTAL_TILEID,L_DIS,R_DIS,MAP
226861024,123.0,0,3,0,35,17,0,"[-1, 0, 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]"
226861024,124.0,1,0,0,35,10,0,"[1, 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]"
226861024,201.0,2,1,0,35,10,15,"[-1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1]"
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, -1, 0, 0, -1, 0, 0, 0, 0, -1, -1, -1, -1, -1, -1, 0]"
```

Figura 6.15: 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 6.16, 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 6.16: Ejemplo de Resultado del histroial.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

Tras la investigación llevada a cabo en este proyecto, los resultados obtenidos, al aplicar redes neuronales convolucionales sobre los datos prestados a la empresa, no han sido los deseados, ya que la mayor precisión obtenida ha sido del 57 %, un valor para nada aceptable.

Es por ello, que creemos que al contar con un conjunto desbalanceado y con, tal vez, pocos ejemplos, los resultados no han sido tan buenos como los que cabría esperar. Además, es probable que este tipo de red neuronal no sea la mejor ante los datos de entrada. Por lo que, consideramos que si se consiguen más datos, y quizás con un nuevo enfoque, se puedan obtener precisiones aceptables por parte de la empresa.

Finalmente, y como es obvio, no se está satisfecho con los resultados obtenidos, aunque pensamos que se han seguido buenas estrategias frente a los datos prestados. Por todo lo anterior, en el siguiente apartado se plantean varias propuestas para mejorar los módelos y conseguir resultados más precisos.

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. **Conocer de manera exacta los criterios de validación de las bobinas:** como ya se ha comentado previamente, la empresa no nos ha dicho en ningún momento cuáles son los criterios exactos que usan para dar como válida o no una bobina, sino que simplemente con los datos medidos nos han pedido que creemos un modelo que sea capaz de aprender y realizar predicciones. Es por ello, que si se supieran sus criterios, se podría ajustar de alguna forma el modelo o utilizar alguna otra herramienta que mejore y facilite las predicciones.

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.

Apéndices

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 *scrum*. Las reuniones se producían, salvo alguna excepción, cada dos semanas donde se producía cada uno de los diferentes *sprints*. En ellos, se revisaban los objetivos marcados en la semana anterior y se marcaban los objetivos del próximo *sprint*.

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/ifh1001/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
Coste total	**870**

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[1]). En la Tabla A.2 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
Sueldo bruto anual	**29.800**

Tabla A.2: Costes personal estimados

[1]Jobted: https://www.jobted.es/salario/programador

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.

Tipo de coste	**Coste en €**
Coste hardware	870
Coste personal	29.800
Coste total	**30.670**

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 también son confidenciales y no pueden ser divulgados públicamente, lo cual supone un problema si quiere utilizar la aplicación alguien que no sea la empresa, ya que, a priori, no contaría con los datos necesarios para usarse.

Tras todo lo anteriomente 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 maracdo los siguientes objetivos generales:

- Ayudar a la empresa que origina el proyecto, consiguiendo que se puedan identificar el mayor número de bobinas con errores aprovechables por parte de la empresa.

- Aumentar la eficiencia de la cadena de galvanizado de la empresa, de manera que un operario no tenga que analizar de forma exhaustiva cada bobina, para saber si es utilizable o no.

- Crear una cadena más sostenible, consiguiendo que se puedan aprovechar al máximo las bobinas galvanizadas y evitar que se tenga que desperdiciar demasiado material.

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 rsultados 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.

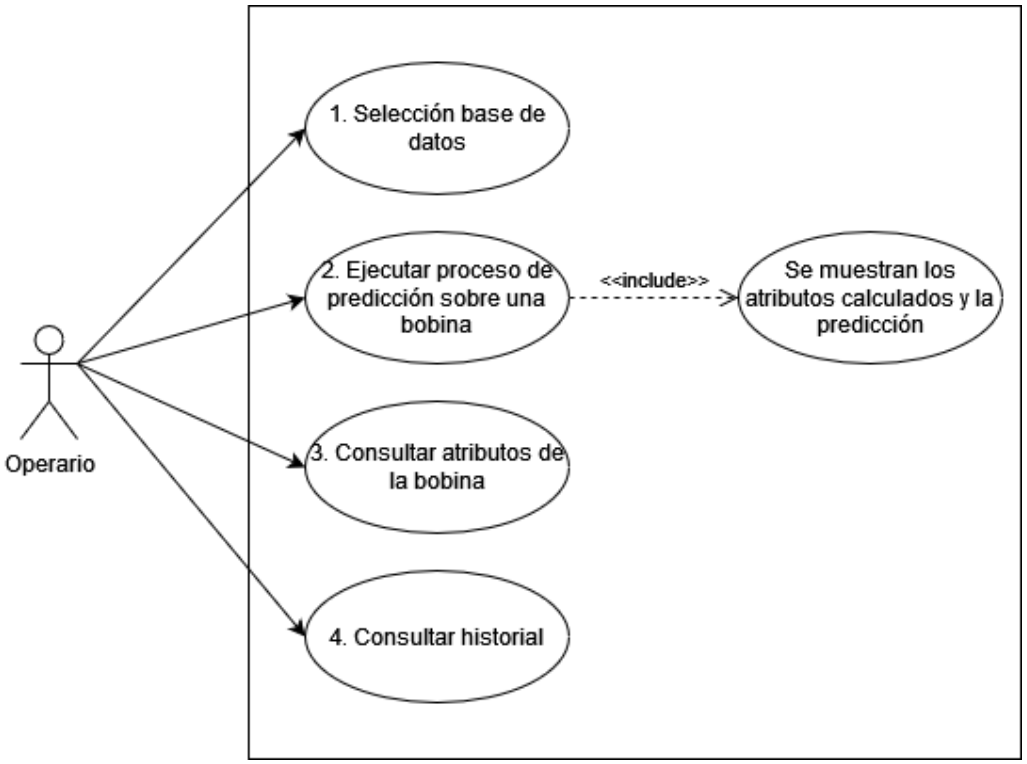

Figura B.1: Diagrama de Casos de Uso

Especificación de casos de uso

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

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

CU 2 - Ejecución proceso de predicción sobre una bobina	
Descripción	Permite al operario seleccionar la ID de la bobina desea sobre la que calcular las predicciones.
Requisitos	REQ 1.2, REQ 1.3 y REQ 1.4
Precondiciones	Se está ejecutando el notebook de la aplicación, y se ha indicado la base de datos.
Secuencia	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.
Postcondiciones	Se muestran los atributos calculados de la bobina y la decisión del modelo.
Excepciones	Se produce algún problema durante el proceso y el modelo devuelve algún error.

Tabla B.2: Caso de Uso - 2

CU 3 - Consultar atributos de la bobina	
Descripción	Permite al operario conocer los atributos y el mapa codificado de la bobina previamente calculados.
Requisitos	REQ 2.1
Precondiciones	Se ha realizado una predicción sobre la bobina deseada.
Secuencia	1. El usuario se dirige a la ruta donde se encuentra la aplicación. 2. El usuario se dirige al directorio /bobinas. 3. El usuario abre el CSV correspondiente a la bobina deseada.
Postcondiciones	El usuario consigue abrir el fichero correctamente, y en su interior se encuentran todos los atributos.
Excepciones	-

Tabla B.3: Caso de Uso - 3

CU 4 - Consultar el historial	
Descripción	Permite al operario poder revisar el historial de las diferentes predicciones realizadas.
Requisitos	REQ 2.2 y REQ 3
Precondiciones	Se ha realizado al menos una predicción sobre cualquier bobina.
Secuencia	1. El usuario se dirige a la ruta donde se encuentra la aplicación. 2. El usuario sabre el archivo historial.txt
Postcondiciones	El archivo contiene en su interior los resultados previamente calculados.
Excepciones	-

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 subpunnto 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 10 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 anteriomente mencionado, de la capa superior.
  - **202:** es la pareja del sensor 201. Toma medida 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.
  - **1243:** es la pareja del sensor 1234, y toma datos de la capa inferior de la bobina.

- **GR_LPValues:** contiene los 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 diferentes ID, su orden es cronológico, es decir, la prima 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 en la tabla *Gauges*.
  - **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. Los atributos usados son los mismos que los de la tabla anterior, aunque hay que aclarar que ahora las tejas funcionan de forma que cada 9 tejas forman una columna, por lo que el orden para componer la bobina será de arriba hacia abajo y de izquierda a derecha, cambiando de columna cada 9 registros.
- **V_Coils:** contiene las bobinas 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.

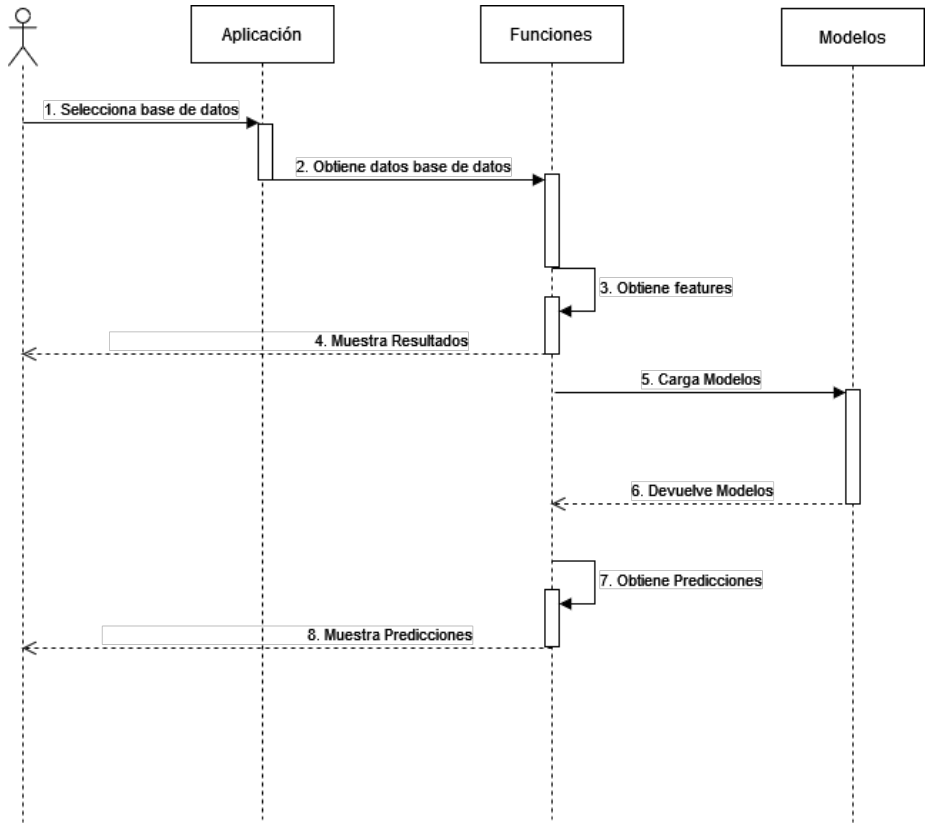

Figura C.1: Diagrama de Secuencia

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 *features* 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[1] 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:

[1]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 LaTeXque 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 él mejore 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 *features*. 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 6 valores, y para cada uno de ellos calcular y representar la curva ROC junto con su AUC, de forma, que se ha trabajado con aquel que mejor AUC tenía, ya que este criterio es él más óptimo para separar las clasificaciones entre una clase u otra.

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 en 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 y se podrá 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 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.

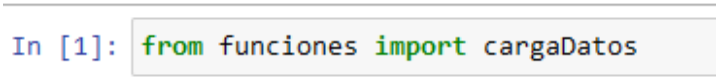

Figura E.1: Ejecución de la Primera Celda

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.

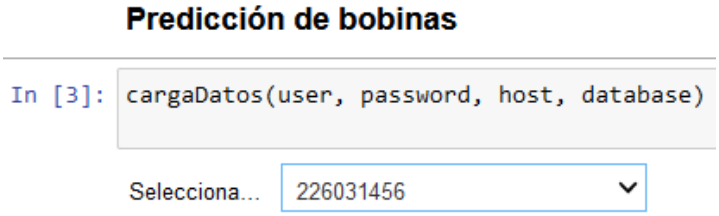

Figura E.2: Ejecución de la Tercera Celda

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 historia.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.

Bibliografía

[1] Wikimedia Commons. File:convolution arithmetic - padding strides odd.gif — wikimedia commons, the free media repository, 2020. [Online; fecha de acceso 27-Junio-2023].

[2] Wikimedia Commons. File:max pooling.png — wikimedia commons, the free media repository, 2021. [Online; fecha de acceso 27-Junio-2023].

[3] Wikimedia Commons. File:roc-draft-xkcd-style.svg — wikimedia commons, the free media repository, 2021. [Online; accessed 28-June-2023].

[4] Ferros Planes. Proceso de galvanizado - ventajas y aplicaciones. https://ferrosplanes.com/proceso-galvanizado-ventajas/, 2018.

[5] 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.

[6] Fernando Izaurieta and Carlos Saavedra. Redes neuronales artificiales. *Departamento de Física, Universidad de Concepción Chile*, 2000.

[7] 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.

[8] 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.

[9] J.A. Martínez Pérez and P.S. Pérez Martin. La curva roc. *Medicina de Familia. SEMERGEN*, 2023.

[10] Na8. Clasificación con datos desbalanceados: principales soluciones. https://www.aprendemachinelearning.com/clasificacion-con-datos-desbalanceados/, 2020.

[11] Juan Sebastian Gelvez Prieto. Redes neuronales convolucionales y redes neuronales recurrentes en la transcripción automática, 2019.

[12] Wikipedias. Curva roc — wikipedia, la enciclopedia libre, 2022. [Internet; descargado 24-junio-2023].