UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU D'ENGINYERIA INFORMÀTICA

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

PREDICTIVE MAINTENANCE USING DEEP LEARNING

Autor: José Manuel López Camuñas

Directors:	**Dr. Simone Balocco**
Tutor Empresa:	**David Pérez Durán**
Realitzat a:	**Departament de Matemàtiques i Informàtica**

Barcelona, **20 de juny de 2021**

Contents

List of Figures

Resumen

El objetivo de este estudio es demostrar que los fallos detectados en un avión pueden estar relacionados con las condiciones ambientales a las que se ve expuesto durante su tiempo de operación. Este estudio es el primer paso de un proyecto de mantenimiento predictivo a largo plazo desarrollado por DMD Solutions.

Primeramente, se introducen los conceptos de fiabilidad y mantenimiento predictivo. A continuación, los fundamentos del aprendizaje automático y el estado del arte.

La obtención de unos datos de calidad ha sido un proceso complejo, ya que los datos disponibles eran incompletos, ruidosos y desequilibrados. Se describen i debaten diferentes soluciones.

Se realizaron dos aproximaciones diferentes: la primera consistía en la predicción de fallos (clasificación binaria) y la segunda, más ambiciosa, la predicción del tiempo restante hasta el siguiente fallo utilizando intervalos de tiempo (clasificación multiclase). Ambas aproximaciones fueron diseñadas utlizando un proceso iterativo que mejoraba la calidad tanto de ambos modelos como de los datos en cada etapa del estudio.

Los resultados obenidos fueron esperanzadores y han animado a una mayor investigación.

Resum

L'objectiu d'aquest estudi és demostrar que les avaries reportades en un avió poden estar relacionades amb les condicions ambientals que està exposat durant el seu temps d'operació. Aquest estudi és el primer pas d'un projecte de manteniment predictiu a llarg termini desenvolupat per DMD Solutions.

En primer lloc, s'introdueixen els conceptes de fiabilitat i manteniment predictiu. A més, els fonaments de l'aprenentatge automàtic i l'estat de l'art.

La recollida d'unes dades de qualitat ha estat un procés complex, ja que les dades disponibles eren incompletes, sorolloses i incloïen alguns desequilibris. Es descriuen i es debaten diferents solucions.

Es van dur a terme dues aproximacions diferents: la primera consistia en la predicció d'avaries (classificació binària), i la segona, més ambiciosa, la predicció del temps restant fins al següent defecte utilitzant intervals de temps (classificació multiclasse). Totes dues aproximacions van ser dissenyades utilitzant un procés iteratiu que millorava la qualitat tant dels models com de les dades en cada etapa de l'estudi.

Els resultats obtinguts van ser esperançadors i han encoratjat una major recerca.

Abstract

The goal of this study is to demonstrate if failures reported in an aircraft can be related to the environmental conditions during operation time. The current study is the first step of a long-term predictive maintenance project driven by the company DMD Solutions.

First of all, the concepts of reliability and predictive maintenance are introduced. Furthermore, the fundamentals of machine learning and the state of the art are detailed.

Gathering quality data was a complex process, since the available data was incomplete, noisy and unbalanced. The analysis proposes and compares several solutions.

Two different approaches were carried out: the first one consisted of the prediction of failure (binary classification), and the second one, more ambitious, the prediction of the time before the next defect using time intervals (multi-class classification). Both approaches were designed using an iterative process that improved quality of both models and data at each stage of the study.

The obtained results were promising and encourage further research.

Agradecimientos

Primero de todo quiero agradecer a mis dos tutores, Dr. Simone Balocco y David Durán por sus consejos, soporte y su gran paciencia durante todo el desarollo de mi trabajo de final de grado. Tambien quiero agradecer a todos los compañeros de DMD Solutions que me han acompañado y ayudado con su experiencia en un campo que para mi era desconocido.

Por último, expresar mi gratitud a mi familia, mis amigos mas cercanos y a la que ha sido mi pareja durante el 90% del estudio. Sin vuestra comprensión, ánimos y cariño, nunca hubiera sido posible completar esta gran tarea.

Chapter 1

Introducción

La industria de la aviación ha sido históricamente asociada a grandes inversiones económicas, y la seguridad, en particular, siempre ha sido una figura clave en estos gastos desde el principio. Cada año, millones de vidas dependen de la fiabilidad de estas grandes máquinas voladoras y por tanto no es de extrañar que en la actualidad haya una gran preocupación sobre temas relacionados con la seguridad dentro de las grandes compañías. La importancia por esta seguridad se encuentra presente en todos los procesos: desde el diseño hasta el mantenimiento. La seguridad pues, no solo juega un papel fundamental a la hora de salvar vidas humanas, sino que también implica grandes costes. Un avión estrellado representa una gran pérdida de materiales y una pérdida en la reputación que puede llevar a desencadenar resultados financieros inesperados a largo plazo.
El concepto de seguridad se encuentra altamente relacionado con el de fiabilidad. La fiabilidad se define como la probabilidad de un sistema (o un elemento de este) funcione correctamente para la tarea que ha sido diseñado bajo unas condiciones concretas y sin fallar durante un periodo de tiempo determinado. Por otro lado, la mantenibilidad hace referencia a la facilidad con la que se pueden realizar las tareas de mantenimiento de un equipo. Su propósito es medir la probabilidad de que un elemento en mal estado pueda ser restaurada a la normalidad después del mantenimiento. La fiabilidad también es cercana a la mantenibilidad, ya que se pueden programar tareas de mantenimiento para asegurar unas condiciones óptimas y extender el tiempo de vida del equipo. Finalmente, la disponibilidad juega su propio rol. Es importante que un elemento se encuentre disponible todo el tiempo posible para ser usado. Por esta razón, el mantenimiento debe ser realizado a tiempo para evitar fallos mayores que requieran de más tiempo de reparación. Por tanto, RAMS (Reliability, Availability, Maintainability and Safety) es crucial en la aviación donde un pequeño error puede llevar a un coste enorme.
Los equipos tienen un precio enorme, pero eso es solo el principio ya que el precio del mantenimiento suele exceder el precio del propio equipo dependiendo del sector. En el mundo de la aviación se estima que el coste del mantenimiento es aproximadamente un 50-60% del coste operativo total de una aeronave. Para minimizar estos gastos, es crucial tener un buen plan de mantenimiento y detectar los defectos cuanto antes. Un equipo que se encuentra expuesto a un defecto de manera prolongada puede resultar en aún peores consecuencias para la aeronave y por tanto más costes.

En resumen, los costes de mantenimiento se pueden reducir con una planificación eficaz y una detección temprana. Por ello, el Mantenimiento Predictivo (PdM) puede ayudar a mejorar la planificación detectando posibles defectos antes de que sucedan. El auge del análisis de datos, así como del Deep Learning dan paso al concepto de un Mantenimiento Predictivo basado en estas técnicas. Tenemos la oportunidad de procesar una gran cantidad de datos históricos e intentar predecir de manera precisa un posible fallo. DMD solutions, como empresa experta en RAMS ha decidido invertir su tiempo y recursos para implementar el PdM en su propia herramienta RAMS llamada Robin RAMS[3]. Este estudio es el primer paso para un proyecto a largo plazo: se pretende comprobar si es posible predecir defectos analizando los datos de vuelo y las condiciones ambientales.

1.1 Objetivos

El objetivo de este proyecto es analizar si es posible encontrar relaciones entre los datos de los fallos de una aeronave, sus datos de vuelo (ruta, velocidad, altura, etc) y los datos meteorológicos de dichos vuelos, para así predecir los fallos que son debidos a estas condiciones externas. Para llevar a cabo este objetivo se han creado distintas tareas:

- Obtener conocimientos en RAMS para poder entender mejor el problema.
- Analizar los datos de defectos (FRACAS).
- Obtener los datos de vuelos asociados a los defectos.
- Obtener los datos meteorológicos asociados a los datos de vuelo.
- Estudiar y relacionar los datos obtenidos.
- Creación de un modelo de predicción.
- Analizar los resultados y extraer conclusiones.

1.2 Justificación del proyecto

En el mundo de la aviación, cada defecto genera grandes pérdidas, incluso aunque al final el defecto fuera una falsa alarma. Sin embargo, estas pérdidas serían aún mayores si el defecto no hubiera sido detectado prematuramente. Si el avión sigue circulando con un defecto, las consecuencias finales pueden ser peores y por tanto el coste de reparación mayor. Cuando se realiza una tarea de mantenimiento después de la detección de un fallo, estamos ante un Mantenimiento Reactivo. Para reducir la fatalidad de los defectos, es importante realizar un mantenimiento regular de la aeronave y así asegurar la detección temprana de defectos. De esta forma, el Mantenimiento Preventivo es llevado a cabo: un mantenimiento basado en la edad de los componentes, y que una vez cierta edad es alcanzada, la pieza debe revisarse. La edad seleccionada para dicho mantenimiento se computa basada en diferentes pruebas realizados durante el propio desarrollo del componente. Este tipo de mantenimiento es mejor que el reactivo ya que el defecto es detectado antes y sus consecuencias se ven reducidas. Este tipo de Mantenimiento Preventivo es

realmente útil cuando hay una gran relación entre la probabilidad de fallo y la edad del equipo, pero para otros muchos casos puede llevar a un mantenimiento repetitivo, prematuro e innecesario. Para anticipar los defectos y evitar mantenimientos innecesarios, nace el Mantenimiento Predictivo. Este tipo de mantenimiento está basado en los datos actuales del vuelo y los datos históricos. Con estos datos, los defectos pueden ser estudiados e intentar averiguar qué los causa. Por ello, en este proyecto, estudiaremos que condiciones generan estos defectos. Si los resultados obtenidos concuerdan con lo esperado, en el futuro, será posible trabajar con el sistema interno de datos de la aeronave o Health and Usage Monitoring System (HUMS).

1.3 Planificació del proyecto

Previo al desarrollo del proyecto, se planificaron diferentes tareas y se estimó su duración para tener las ideas más organizadas, asignar plazos de entrega a cada tarea y mejorar la visualización del proceso. A continuación, se presenta el gantt inicial optimista.

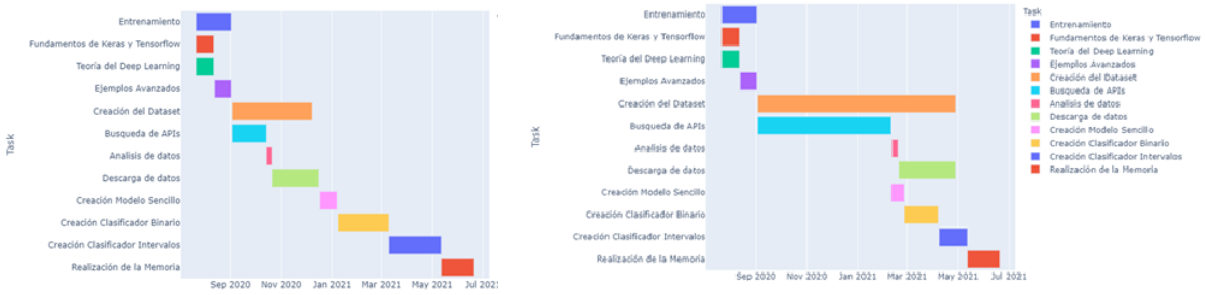

Figure 1.1: Tiempo estimado vs Tiempo real

Finalmente tenemos otro gantt que representa la distribución real del tiempo. Podemos notar como las etapas de entrenamiento coinciden, pero la creación del dataset y la búsqueda de fuentes de datos fiables tuvieron una mayor durada a la planeada debido a dificultades (limitaciones en número de descargas, velocidad de bajada, etc). Esto nos dejó con menos tiempo para la etapa del modelo, pero aun así se pudieron realizar satisfactoriamente, respetando el plazo de entrega.

Chapter 2

Deep Learning

Los términos Inteligencia Artificial, *Machine Learning* y *Deep Learning* suelen utilizarse en el mismo contexto para describir software que se comporta de manera inteligente. Sin embargo, es importante entender que dichos términos presentan diferencias entre ellos.

Podemos pensar en el *Deep Learning*, *Machine Learning* e Inteligencia Artificial como subconjuntos que forman parte de conjuntos más grandes (a mayor es el conjunto, menos concreto se vuelve el término). Siguiendo esta lógica, el *Deep Learning* es un subconjunto del *Machine Learning* y el *Machine Learning* es un subconjunto de la Inteligencia Artificial. En otras palabras, todo el *Machine Learning* forma parte de la Inteligencia Artificial, pero no toda la Inteligencia Artificial es *Machine Learning*.

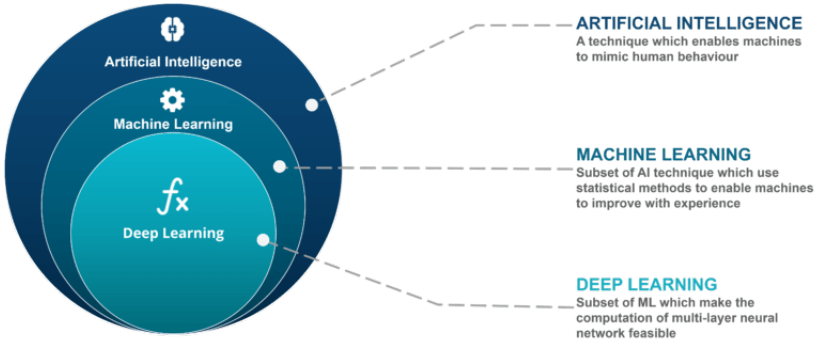

Figure 2.1: Jerarquía de términos

Como vemos en la anterior figura, el *Deep Learning* se caracteriza por el uso de Redes Neuronales.

2.1 Deep Feedforward Networks

Este tipo de redes son la quintaesencia del *Deep Learning*. La meta es aproximar una función f^*. Por ejemplo, un clasificador, $y = f^*(x)$ mapea un input x a una categoría y.

En resumen, definen un mapeo $y = f(x;\alpha)$ y aprende el valor de los parámetros α que aproximan mejor la función. Son llamadas redes ya que representan una composición de funciones donde cada función se le atribuye a una capa. De hecho, se conoce como *Deep Learning* debido a que mayor número de capas, mayor es la profundidad de la red. [26]

Finalmente, son llamadas neuronales porque están altamente inspiradas en la neurociencia. Cada capa está representada por un vector y cada elemento del vector constituye una neurona. La dimensionalidad del vector representa la anchura del modelo.

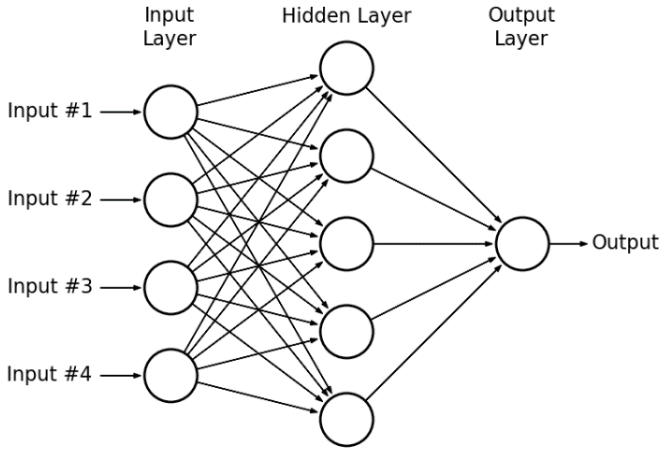

Figure 2.2: Estructura de una Red Neuronal

2.2 Recurrent Neural Networks

Las Redes Neuronales Recurrentes (RNN) son una familia de redes neuronales especializadas en el proceso de datos secuenciales. Se toma ventaja de la idea de compartir parámetros en diferentes partes del modelo. Este concepto hace posible extender el modelo a ejemplos de diferentes formas y generalizar entorno a ellos. Si tuviéramos valores independientes para cada valor temporal no podríamos generalizar para secuencias de un tamaño diferente al visto en la fase de entrenamiento.

Para simplificar, nos referimos a las RNN como una operación sobre una secuencia de vectores $x(t)$ donde t representa el índice temporal (de 1 a t). Estas redes operan por *batch*es de secuencias con una longitud diferente para cada miembro del *batch*. La red tiene conexiones hacia atrás haciendo posible una retroalimentación.

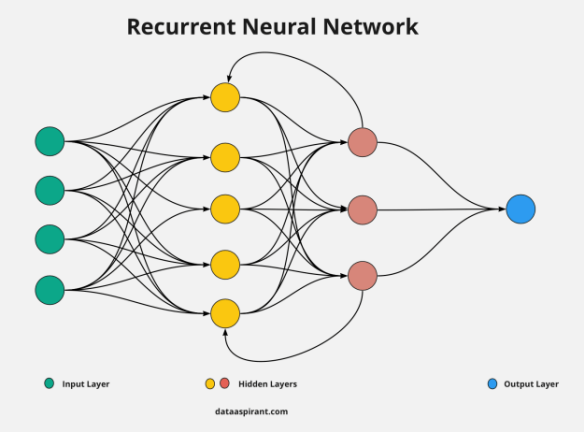

Figure 2.3: Estructura de una Red Neuronal Recurrente

2.2.1 Long Short-Term Memory

Las Redes Neuronales de tipo LSTM son RNN que contienen unidades LSTM. Una unidad LSTM está compuesta por una celda, una puerta de entrada, una puerta de salida y una puerta de olvido. La celda recuerda valores durante un cierto intervalo de tiempo y las otras 3 puertas regulan el paso de la información dentro y fuera de la celda. Mejoran las RNN tradicionales eliminando problemas encontrados como el desvanecimiento del gradiente y también añaden ventajas como la insensibilidad a grandes diferencias de longitud.

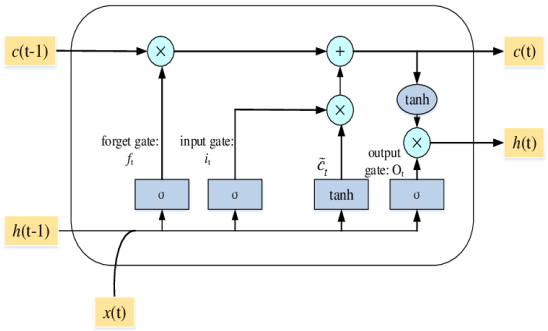

Figure 2.4: Estructura de una unidad LSTM

2.3 Convolutional Neural Networks

Las Redes Neuronales Convolucionales (CNN) son un tipo de red neuronal especializada en el procesamiento de datos que tienen una topología de *grid*. El ejemplo más directo son las imágenes que pueden ser vistas como un *grid* 2D de píxeles. Se llaman convolucionales debido a que emplean una operación lineal llamada convolución.

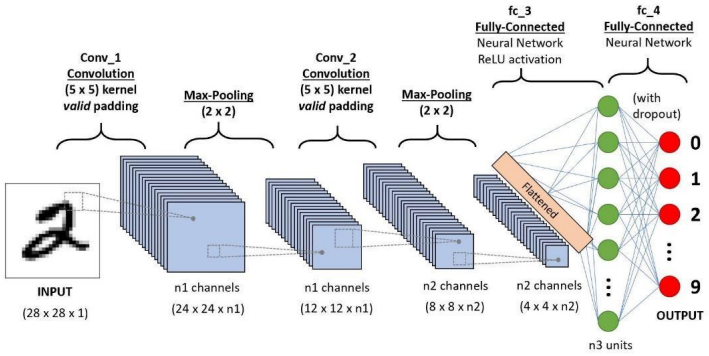

Figure 2.5: Estructura de una Red Neuronal Convolucional

2.4 Layers

Como se ha visto, las Redes Neuronales consisten en un input, una sucesión de *layers* o capas y una capa final de salida. A continuación, vamos a explicar diferentes capas que son de interés para el estudio.

2.4.1 Fully-Connected Layer

Son esas capas donde todos los inputs de una capa están conectados con cada unidad de activación de la siguiente capa. Son las encargadas de compilar los datos extraídos de las demás capas (o los datos que reciben). En las CNN acostumbran a ser las últimas capas, compilando así toda la información que reciben de las capas convolucionales.

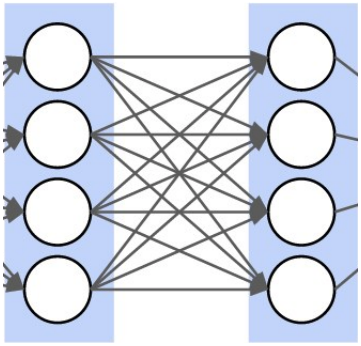

Figure 2.6: Ejemplo de una Fully-Connected Layer

2.4.2 Convolutional Layer

Una capa convolucional consiste en conjunto de pequeños filtros que se aplican respecto el *input* original o respecto otra característica extraída por otro filtro. Este tipo de capas extraen patrones y aprenden a detectarlos por todo el *input*. Además, pueden aprender

una jerarquía de patrones donde los primeros patrones extraídos forman patrones más complejos y así sucesivamente.

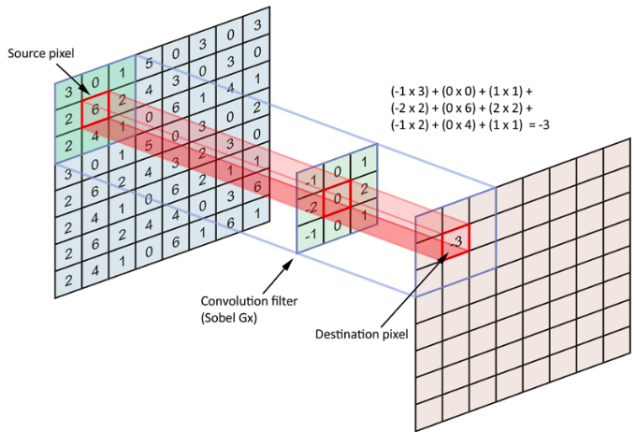

Figure 2.7: Ejemplo de una Convolutional Layer

2.4.3 Pooling Layer

Conocida como capa de *subsampling*, reduce las dimensiones de las características extraídas por las capas de convolución y así reducir el número de parámetros a aprender por la red. Sin ellas el número de características extraídas a aprender sería mucho mayor.

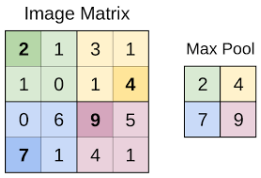

Figure 2.8: Ejemplo de Max Pooling

2.4.4 Dropout Layer

Esta capa asigna 0 la salida de una neurona con una determinada frecuencia r añadiendo ruido y obligando a las neuronas a tomar más responsabilidad con el *input*. Esto ayuda a prevenir el *overfitting*.

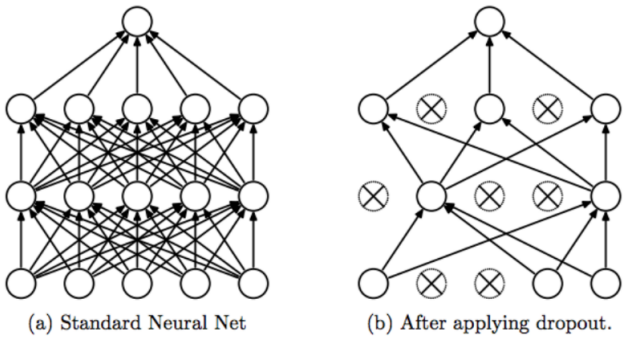

Figure 2.9: Ejemplo de Dropout

2.4.5 Normalization Layer

Este tipo de capas pretenden normalizar los valores de entrada centrando la media en 0 y otorgándole una desviación estándar unitaria. Esto se logra restando los valores con la media y dividiendo los datos por su propia desviación estándar. Esta capa permite llevar a cabo la normalización de cada *batch* durante el entrenamiento.

La normalización puede llevar a reducir notablemente el tiempo de entrenamiento necesario para una red.

2.4.6 Activation Layer

La capa de activación decide el valor final de cada neurona. Permiten tanto un funcionamiento lineal como no lineal dependiendo de la función usada. Algunas de ellas son:

- ***Rectified Linear Unit* (ReLu)**:Elimina los valores negativos transformándolos en 0.

$$ReLu(x) = max(0, x) \tag{2.1}$$

- ***Softmax***: Limita el rango de los valores de un vector entero entre 0 y 1. La salida representa la distribución categórica o distribución de probabilidad sobre las diferentes posibles salidas del vector.

$$Softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}} \tag{2.2}$$

- ***Sigmoid***: Limita el rango de los valores entre 0 y 1. Suele ser usado para el cálculo de probabilidades individuales.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

2.5 Funciones de Loss

La función de Loss permite evaluar nuestro modelo indicando su error y permitiendo al optimizador calcular los mejores parámetros que reducen el error producido por esta función. La selección de una función de Loss adecuada al problema es extremadamente importante para obtener buenos resultados.

El estudio ha contemplado principalmente las siguientes funciones:

- ***Mean Squared Error* (MSE)**: Mide el error cuadrado promedio. Para cada punto se calcula la diferencia respecto el valor a predecir y esta se eleva al cuadrado para finalmente calcular el promedio de estas.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.4}$$

- ***Root Mean Squared Error* (RMSE)**: Es la raíz cuadrada del MSE. Es útil cuando se quieren penalizar mucho los errores muy grandes (en contraste al MSE).

$$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} \tag{2.5}$$

- ***Cross-Entropy* (CE)**: Calcula la diferencia o error entre 2 distribuciones distintas.

$$CE = -\sum_{i=1}^{N} q(y_i) \cdot log(p(y_i)) \tag{2.6}$$

  q(y) es la distribución real, *p(y)* es la distribución calculada y N es el número de clases.

2.6 Optimizadores

Definen la forma en que el modelo actualizará sus valores basándose en función de *Loss* para tratar de minimizar el valor de dicha función. Durante este estudio hemos observado los siguientes optimizadores:

- ***Stochastic Gradient Descent* (SGD)**: El descenso del gradiente es un algoritmo iterativo que pretende encontrar el mínimo local 0 de una función. Para ello nos desplazamos en la dirección del gradiente (usando todos los datos) y controlando la distancia de este desplazamiento mediante el *learning rate*. El SGD aproxima este comportamiento usando un subconjunto de los datos para computar el gradiente. Tiene variaciones como el *Momentum* SGD o el *Nesterov* SGD que aceleran este descenso cuando la dirección de descenso se mantiene similar entre cada desplazamiento.

- ***Root Mean Square Propagation* (RMSProp)**: Es otra aproximación del descenso del gradiente que en vez de usar un *learning rate* fijo, se va modificando con cada paso dividiendo este valor por la media del declive exponencial del cuadrado de los gradientes.
- ***Adaptative Moment Estimation* (ADAM)**: Mantiene un *learning rate* por parámetro y además de calcular el RMSProp, este valor se modifica con la media del *momentum* del gradiente.

Chapter 3

Estado del Arte

3.1 Reliability

La fiabilidad o *reliability* es una característica de un objeto que expresa la probabilidad de que dicho objeto realice la función para la que ha sido diseñado en unas determinadas condiciones y durante un intervalo de tiempo establecido. Generalmente se representa como *R*. Desde un punto de vista cualitativo, la *reliability* también puede verse definida como la habilidad del objeto para seguir siendo funcional. Cuantitativamente, la *reliability* especifica la probabilidad de que no habrá una parada operacional durante el intervalo de tiempo establecido. Por tanto, a más fiable sea un componente, menos probable será que falle durante la operación [22].

3.1.1 FRACAS: Reliability basada en datos reales

FRACAS (*Failure Reporting Analysis, and Corrective Action System*) es una de las metodologías más extensamente usadas para el análisis de fiabilidad en la industria aeroespacial. FRACAS proporciona una estructura de proceso para el cálculo de los parámetros de fiabilidad (como el *Mean Time Between Failures* o *MTBF*) en base a los datos reales de operación del sistema. FRACAS es una herramienta imprescindible para todo el ciclo de vida, desde el prototipado hasta la venta. No es solo una herramienta de ingeniería para ganar conocimiento sobre el comportamiento del sistema, sino que además es un requisito obligatorio para obtener la certificación de la Agencia de Seguridad Aérea de la Unión Europea (EASA). El objetivo del proceso FRACAS es planificar, implementar y hacer un seguimiento de las medidas correctivas en respuesta a los fallos que se están analizando.

FRACAS es un sistema que proporciona un proceso para reportar, clasificar, analizar fallos y planear las medidas correctivas en respuestas a esos fallos. Un correcto uso de este sistema puede ayudar a mejorar la calidad y reducir los costes.

El proceso de FRACAS consta de tres etapas:

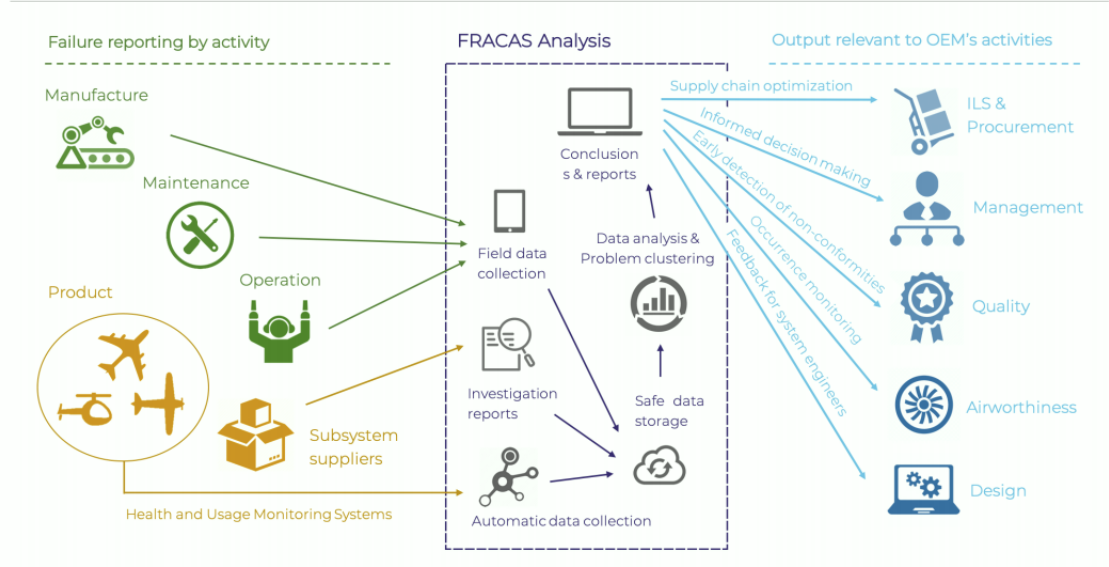

Figure 3.1: Proceso FRACAS [3]

Información de Aeronaves	Type	Tipo de aeronave (fabricante, modelo, etc)
	Serial Number	Número de serie otorgado por el fabricante.
	Flying Hours	Horas acumuladas de vuelo cuando sucede un fallo.
Información de Partes	Part Number	Identificador de la parte.
	Serial Number	Número de serie otorgado por el fabricante.
	ATA Chapter	Identifica donde se encuentra la parte.
	Flying Hours	Horas de vuelo acumulados cuando sucede el fallo.
Información de Defectos	Defect Date	Fecha del defecto.
	Description	Descripción de como se detectó el error y sus consecuencias.
	Defect Type	Categorización del error.
	Finding	Explica la causa del defecto y la corrección a realizar.

3.1.1.1 Failure Report

Para realizar un FRACAS fiable, es crucial reportar todos los defectos ocurridos en una aeronave. Todos los defectos deben estar guardados en una base de datos sólida para poder hacer realizar un correcto seguimiento y análisis. Esta parte requiere de la participación del usuario del componente (por ejemplo, la aerolínea que comunica el defecto) que debe detallar tanto como sea posible las condiciones en el momento del fallo. La siguiente tabla consta de los parámetros de entrada listados por FRACAS.

En este proyecto, la categorización seguida para los defectos ha sido la siguiente:

- ***Defect Verified* (DV):** Asignado cuando el fallo ha sido confirmado.
- ***Open*:** El defecto no ha sido investigado aÚn. Generalmente, hay una política fijada por la compañía que computa estos defectos como DV. (por ejemplo, el 90% de Open

son DV).

- ***Not Fault Found* (NFF):** Asignados cuando la parte ha sido probada, no se ha encontrado ningún fallo y la parte funciona correctamente.
- ***Discard* (DSR):** Asignado cuando una parte que no funciona correctamente no puede ser reparado y es directamente extraída sin ningún análisis adicional.
- ***Scheduled Maintenance* (SCM):** Asignado cuando una parte es extraída para un Mantenimiento Programado.
- ***Scheduled Service Bulletin* (SBS):** Asignado cuando la parte es extraída para realizar una modificación llamada *Service Bulletin*.
- ***No Defect* (ND):** Asignado cuando un defecto no es atribuible a la propia unidad, pero si a un factor externo que puede ser del entorno (por ejemplo, un relámpago, el choque con algún pájaro), un error humano o un mantenimiento mal realizado. Las extracciones debido al desgaste de consumibles como los neumáticos también son considerados como ND.

3.1.1.2 Analysis

El segundo paso consta del análisis de los fallos reportados. Usando toda la base de datos, los *reliability assessment* (evaluaciones de la fiabilidad) son usados comúnmente para valorar la fiabilidad de las diferentes partes. Estas valoraciones se pueden realizar a diferentes niveles:

- ***Part Level*:** Solo las partes de la aeronave son analizadas (por ejemplo, un actuador).
- ***System Level*:** Un sistema entero de la aeronave es analizado (por ejemplo, el sistema eléctrico).
- ***Aircraft Level*:** Se analiza la aeronave entera (todos los componentes).

Las horas acumuladas de vuelo de toda una flota entera deben tenerse en cuenta también. Este valor puede ser obtenido de los diferentes informes realizados por el operador de la aeronave.

Para monitorizar la fiabilidad, varios *Key Parameter Indicator* (KPI)s son usados. Todos ellos están basados en las horas totales de vuelo de la aeronave. Los más comunes son:

- ***Failure Rate* (λ):** Este indicador muestra cuántas veces fallará un objeto por hora. Normalmente, el orden de magnitud es 10^{-6}, así que se expresa en fallos por millón de horas (fmph).
- ***Mean Time Between Failures* (MTBF):** Este parámetro indica el tiempo promedio entre que dos fallos son reportados para un mismo ítem. Es el concepto inverso de *failure rate*. La constante λ normalmente tiene la forma de un *bath-tube graph*. El MTBF es usado para sistemas reparables, mientras que el *Mean Time To Failure* (MTTF) denota el tiempo esperado al fallo para un sistema no reparable.

$$\lambda = \frac{1}{MTBF} = \frac{1}{MTTF} \tag{3.1}$$

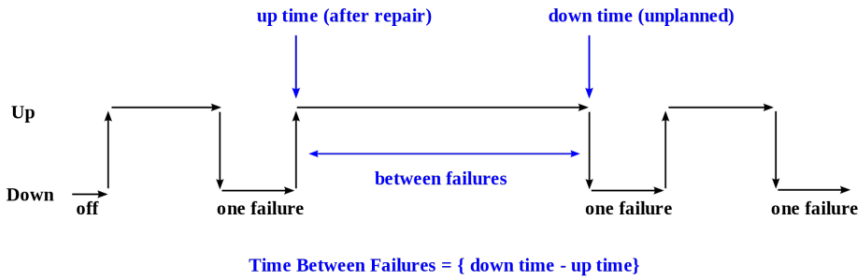

Figure 3.2:
Time Between Failures [2]

Este parémetro sólo considera esas extracciones donde ha surgido un mal funcionamiento inherente al ítem (DV, DSR, *Requested* DV, *Open* y un factor f que simboliza el ratio de defectos *Open* que se consideran DV).

$$MTBF = \frac{\sum i_{aircraft} FHi_{aircraft}}{\sharp DV + \sharp DSR + \sharp RDV + f \times \sharp Open} \tag{3.2}$$

***Mean Time Between Unscheduled Removals* (MTBUR):** Este parámetro indica el tiempo medio entre dos extracciones no programadas para un mismo ítem (DV, DSR, RDV, ND, NFF, *Requested* NFF, *Requested* DSR, *Open* y el factor f).

$$MTBUR = \frac{\sum i_{aircraft} FHi_{aircraft}}{\sharp DV + \sharp DSR + \sharp RDV + \sharp ND + \sharp NFF + \sharp RNFF + \sharp RDSR + f \times \sharp Open} \tag{3.3}$$

Los *reliability assessments* computan estos KPIs y su evolución histórica en los meses previos. Estos KPIs son discutidos en meetings llamados *Reliability Review Board* (RRB) y el *reliability team* puede comparar los resultados de las valoraciones y comprobar si los resultados son satisfactorios. Por norma general, estos KPIs son comparados con el MTBF garantizado por el proveedor (GMTBF). En caso de notar un rendimiento peor al indicado, se contacta con el proveedor y se define una medida correctiva que mejore dicho rendimiento.

3.1.1.3 Corrective Actions

Con los resultados del análisis, podemos definir unas medidas correctivas. Las medidas correctivas responden a los problemas detectados en el análisis. Estos problemas pueden estar relacionados con el rendimiento de un ítem, mal registro del mantenimiento, etc. Para ilustrar este paso, mostramos la siguiente figura.

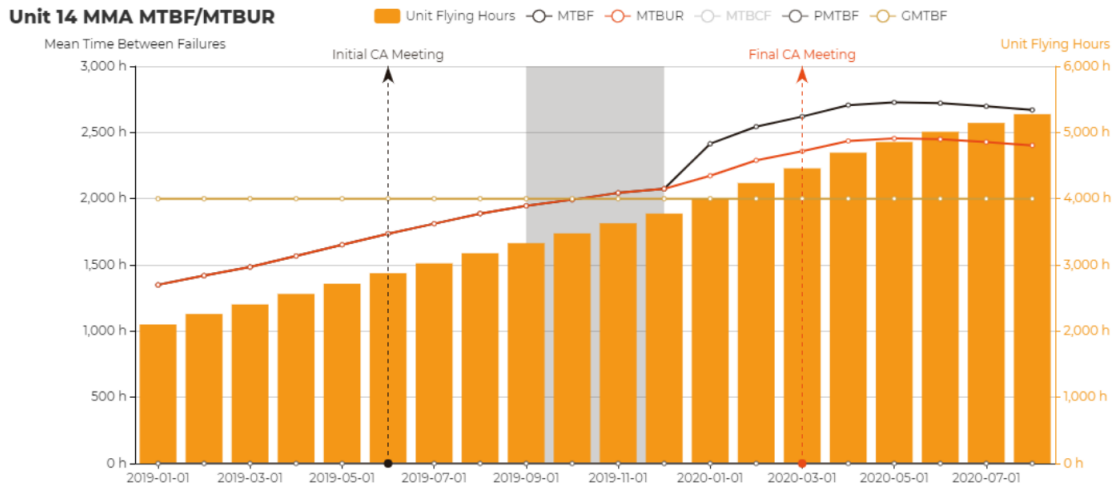

Figure 3.3: Evolución del MTBF y del MTBUR durante la implementación de una medida correctiva (CA) [3]

Como podemos ver, la unidad tenía un MTBF (línea negra) por debajo del GMTBF (línea amarilla), por tanto, el fabricante decide aplicar una medida correctiva. Después de aplicar dicha medida, notamos una mejora del MTBF e incluso hay algunos meses donde la pieza no sufre ninguna avería (sección en gris). El fabricante finaliza la medida correctiva cuando nota que el MTBF es mayor que el GMTBF.

3.1.2 Reliability Prediction Analysis (RPA)

Como hemos visto en la anterior sección, FRACAS nos proporciona forma de predecir la *reliability* de los componentes trabajando con datos reales y actuales, pero existen otros métodos que tratan de hacer lo mismo. Este tipo de predicciones alternativas son usadas durante la fase de diseño para obtener los valores de fiabilidad y realizar una correcta certificación del producto [17].

3.1.2.1 Métodos teóricos

Con el paso del tiempo la importancia de la fiabilidad ha sido reconocida. Por ello, diferentes entidades se han dedicado a estudiar su comportamiento para intentar realizar predicciones. Esos esos estudios, especialmente en el campo militar, han surgido diversas metodologías predictivas.

3.1.2.1.1 Estándar Militar MIL-HDBK-217F

El estándar *MIL-HDBK-217F – Military Handbook: Reliability Prediction of Electronic Equipment* es un estándar de defensa publicado por el Departamento de Defensa de los Estados Unidos. Fue creado en 1961 y ha sido revisado varias veces. En 1995 fue reescrito por última vez y se completó el estándar. Esta metodología fue principalmente usada para

predecir la fiabilidad de los sistemas electrónicos militares, pero con el tiempo, se ha ido extendiendo a muchos otros campos. Es el estándar más usado pero sus valores de carácter conservativo han hecho que otros estándares más modernos se usen cada vez con más frecuencia.

Los parámetros introducidos por el estándar y usados para calcular el *failure rate* han sido obtenidos del estudio de fallos reales en el campo. *MIL-HDBK-217F* nos otorga un *failure rate* para diferentes componentes electrónicos basados en los siguientes parámetros, los cuales suelen resultar en una multiplicación de factores que representan diferentes tipos de estrés:

- *Failure rate* base del componente (π_{Base}).
- Factor de estrés de la temperatura (π_T).
- Factor de potencia (π_R)
- Factor de voltaje (π_S)
- Factor de calidad (π_Q)
- Factor ambiental (π_E)

Los parámetros necesarios varían según el componente, pero generalmente el cómputo del *failure rate* queda definido por la siguiente ecuación:

$$\lambda(t) = \lambda_{base} \cdot \pi_T \cdot \pi_R \cdot \pi_S \cdot \pi_Q \cdot \pi_E \tag{3.4}$$

3.1.2.1.2 FIDES Guide 2009 version A

FIDES Guide 2009 version A [18] es una guía de metodología global electrónica para la fiabilidad que fue producida por el grupo FIDES bajo la supervisión de la *Direction Générale de l'Armamente* (DGA).

El modelo FIDES considera el calendario *Failure In Time* (FIT). El calendario FIT, de acuerdo con FIDES, es una mejor forma de describir la fiabilidad que el *failure rate per hour operation*, porque el calendario FIT tiene en cuenta no solo las horas de operación, sino que también las que no. Esta aproximación es útil cuando tenemos un ítem que conocemos bien, además se convierte también en un indicador del perfil de vida. Cuando se realiza una predicción con FIDES, es muy importante definir correctamente el perfil de vida del sistema y cada una de las fases que forman parte de este perfil. De lo contrario, las ventajas del calendario FIT se ven reducidas significativamente.

La estimación del *failure rate* puede ser calculada como:

$$\lambda = \lambda_{fisico} \cdot \pi_{PM} \cdot \pi_{process} \tag{3.5}$$

Donde λ_{fisico} es una compilación de varios factores secundarios (impacto térmico. eléctrico, ciclos de temperatura, mecánicos, humedad y condiciones químicas), π_{PM} define la calidad del ítem (PM es la abreviación de *Part Manufacturing*) y $\pi_{process}$ es un símbolo de la calidad y el control técnico sobre la fiabilidad en el ciclo de vida del producto. Proporciona una calificación sobre la madurez del fabricante en el control de su proceso de fiabilidad. Estos valores son computados después de responder diversas preguntas sobre el producto y la misión [19].

3.1.2.2 Métodos basados en datos históricos

Este tipo de métodos registra los datos de fallos y las horas de operación de los diferentes ítems durante los últimos años (o incluso décadas) en diferentes disciplinas (Comerciales y Militares). Estos registros son convertidos en parámetros de fiabilidad como el *failure rate* y son guardados en una gran base de datos. Las fuentes de esta base de datos:

- Informes y *papers* publicados.
- Datos recogidos por estudios del gobierno.
- Datos recogidos por los sistemas de recolección de datos de mantenimiento militar.
- Datos recogidos de los sistemas de reparación de garantías comerciales.
- Datos recogidos de las bases de datos de mantenimiento comerciales/industriales.
- Datos suministrados directamente por organismos militares o organizaciones comerciales que mantienen una base de datos de los fallos.

No es factible tener modelos de *failure rate* para cada tipo de componente o montaje con el estándar *MIL-HDBK-217F* u otras metodologías de predicción. Tradicionalmente, los modelos de predicción han sido aplicables solo para componentes electrónicos genéricos. Por tanto, estas bases de datos cubren una gran variedad de necesidades:

- Proporcionar fuentes de datos de *failure rate* sustitutas cuando una organización no tiene datos de su propio producto.
- Proporcionar *failure rate* en los ensamblajes (por ejemplo, en unidades de disco) en los casos en que los análisis a nivel de parte no son factibles o requeridos.
- Complementar las metodologías de predicción de la fiabilidad proporcionando datos de tipos que no abordan otros modelos.

3.1.2.2.1 EPRD 2014: Electronic Parts Reliability Data

Creado por *Quanterion Solutions Incorporated*. Contiene datos de *failures rates* de campos para componentes electrónicos comerciales y militares para usarlos en aná lisis de fiabilidad. Los tipos de componentes incluyen circuitos integrados, semiconductores discretos, resistencias, condensadores, inductores y transformadores.

3.1.2.2.2 NPRD: Non-Electronic Parts Reliability Data

La base de datos *NPRD-2016* también fue creada por *Quanterion Solutions Incorporated (Quanterion 2016d)*. Esta base de datos contiene datos de *failures rates* para una amplia variedad de componentes de tipo mecánico, electromecánico y ensamblajes electrónicos. Los datos fueron obtenidos de observaciones prolongadas de sistemas y elementos en el campo. La base de datos existe tanto en formato *paper* como electrónico, pero normalmente se adquiere como parte de un producto de *software* en el área de la fiabilidad. En la siguiente figura vemos una captura del NPRD del módulo RPA de *Robin RAMS*.

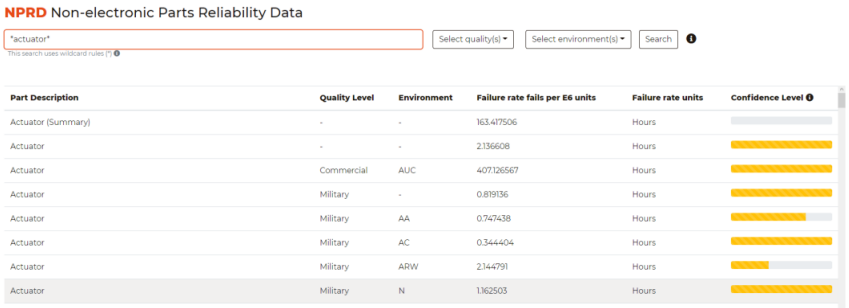

Figure 3.4: Tabla NPRD de Robin RAMS [3]

3.2 Mantenimiento Predictivo

La premisa para el Mantenimiento Predictivo es que monitorizar de manera regular la condición mecánica, eficiencia operativa, y otros indicadores de la condición operativa del tren de máquinas y los sistemas de proceso permitirán proveer de los datos necesarios para asegurar el máximo tiempo entre reparaciones y minimizar los costes [4].

Un programa completo de gestión de Mantenimiento Predictivo utiliza las herramientas más coste-efectivas para obtener la condición operativa real de los sistemas más críticos y programar todas las actividades de mantenimiento a necesidad. También se pretende optimizar la disponibilidad de la maquinaria de proceso, reducir los costes de mantenimiento, mejorar la calidad del producto, la productividad y la rentabilidad en la fabricación.

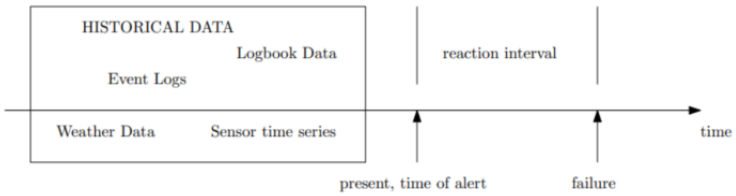

Figure 3.5: El proceso del Mantenimiento Predictivo [4]

Dado el incremento de datos disponibles generados por los procesos de monitorización (sensores y registros de eventos), la meta es predecir los próximos eventos críticos o fallos del sistema. El *machine learning* puede detectar diferentes criterios (condiciones críticas que puedan llevar al fallo) después de analizar la información anteriormente mencionada en relación a los defectos y realizar una predicción [23].

En la actualidad, las aeronaves modernas tienen muchos componentes capaces de registrar su rendimiento. Esto permite guardar una gran cantidad de datos para poder analizarlos. Cada vuelo produce una enorme cantidad de datos que es transmitida a los *OEMs (Original Equipment Manufacture), MROs(Maintenance, Repair and Overhauls)* y aerolíneas. Tradicionalmente, un gran equipo de ingenieros hubiera estudiado los casos y tratado de extraer algunas correlaciones de los datos recolectados y los defectos informados. El *machine learning* implica un cambio fundamental: la máquina puede hacer no solo esta difícil tarea sola, sino que también encontrar correlaciones que un humano normalmente no sería capaz de ver y dejando al humano la evaluación de los resultados que genera la máquina (supervisar que el razonamiento que plantea tenga sentido). Esta es la principal ventaja del Mantenimiento Predictivo [24].

Como se ha mencionado antes, el Mantenimiento Preventivo está basado en intervalos fijos de tiempo definidos por las horas de vuelo o ciclos. Generalmente, el rendimiento se irá deteriorando con el paso del tiempo y el uso. Una de las ganancias con el Mantenimiento Predictivo es que usando los datos puede establecer un determinado *threshold* que sirva de alarma cuando un fallo está por suceder. Se puede tomar la decisión de extraer el ítem prematuramente, con la expectativa de que la reparación será ahora más barata que el reemplazamiento cuando falle.

Sin embargo, no es un proceso tan sencillo o directo ya que durante el desarrollo de este podemos encontrarnos con diversos problemas:

- **Volumen de datos:** para cada vuelo realizado por una aeronave, internamente se generan datos a través de los miles y miles de sensores a bordo. Esto se traduce en un volumen de datos enorme para procesar, limpiar y utilizar. Sin la inversión de equipos que puedan gestionar este volumen, nos encontramos con un problema.
- **Datos no efectivos:** no todos los datos recolectados por los sensores nos interesan o solo una sección de ellos. Es importante filtrar los datos innecesarios para evitar resultados inesperados.
- **Recolección de datos:** por norma general, los datos obtenidos por las aeronaves se encuentran atados a grandes políticas de protección de datos de sus respectivas compañías y solo ellas pueden permitir el acceso. Así pues, es difícil utilizar estos datos en el ámbito de la investigación.
- **Datos incompletos/ruidosos:** algunos sensores fallan o reportan valores que no son reales durante un tiempo determinado. Esto puede generar problemas debido a que los algoritmos de *machine learning* son muy sensibles a este tipo de problemas. Como veremos en este estudio, va a ser difícil tratar con estos datos y recolectarlos de fuentes públicas.

A día de hoy, varias aerolíneas utilizan el Mantenimiento Predictivo para sus flotas de vuelo. Por ejemplo, *Eithad Aviation Group* está trabajando con los fallos detectados en diversos sistemas que pueden ser fácilmente relacionados a las condiciones arenosas sufridas por las aeronaves en su tierra, *Abu Dhabi*. También, *easyJet* ha trabajado estos últimos años en implementarlo en sus planes de mantenimiento diario. Ambas compañías han usado la base de datos *Skywise*, la cual contiene datos de 130 aerolíneas y 9000 aeronaves. *Skywise* fue creada por *Airbus* en 2017 y es una plataforma abierta para la industria de la aviación. Sin embargo, no es una fuente pública de datos y su uso está limitado a las propias aerolíneas, aeropuertos, *MROs* y otros. Como podemos ver en la siguiente figura, en Agosto de 2020 ya había acumulado 12 *PetaBytes* de información.

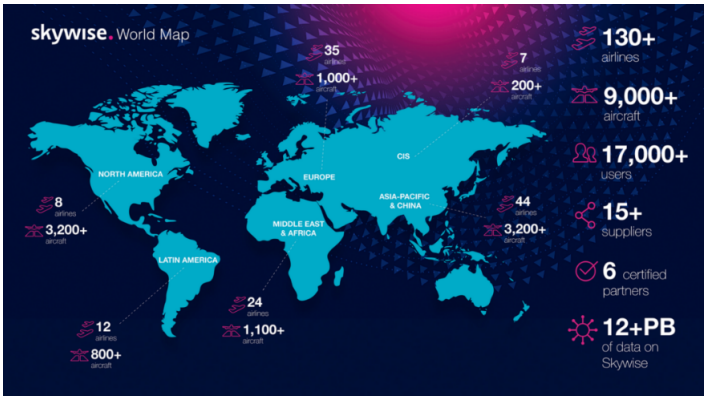

Figure 3.6: Skywise [5]

3.3 Related Work

En esta sección se explora el trabajo previamente hecho por otros en el campo del Mantenimiento Predictivo con *machine learning* para tomar nota de lo que se ha realizado hasta ahora y ver cual es la tendencia.

3.3.1 Clasificación de Series Temporales: Convolutional LSTM

Con el crecimiento del volumen de datos generados por el humano durante el tiempo y la facilidad para guardar estos datos ha nacido un interés en procesar estos datos para entender más su naturaleza y evolución. Estas series temporales vienen caracterizadas por la dependencia temporal a corto y largo plazo. En los últimos años se ha visto un esfuerzo en la creación de modelos que puedan aprender también de estas dependencias temporales y como resultado las Redes Neuronales Recurrentes (RNN) han evolucionado en redes más complejas que sirvan para este propósito. Un ejemplo sería la *Long Short-Term Memory* o LSTM que mejora la capacidad de las RNN básicas.

En el paper *A Hybrid Deep Representation for Time Series Classification and Prediction* se menciona la utilidad de estas para el Mantenimiento Predictivo. Proponen un modelo híbrido

de Red Convolucional Recurrente usando capas LSTM que muestra un buen rendimiento tanto en tareas de clasificación como de predicción. Más *papers* han sido investigados mostrando la gran inmensa mayoría la misma conclusión, el uso de las capas LSTM es el camino a seguir.

Chapter 4

Data Acquisition

En este capítulo se explora el proceso de recolección de los datos, las fuentes donde se han obtenido y cómo se han estructurado. Contamos principalmente con tres fuentes de datos: datos de FRACAS, datos de vuelo y datos meteorológicos.

4.1 FRACAS Data

Extraemos la información de los defectos, así como la identidad de las aeronaves de FRACAS. El fabricante que estudiamos utiliza jets ligeros privados y aviones de entrenamiento militar. FRACAS toma los datos de la flota entera, las horas de vuelo mensuales y todos los defectos reportados del fabricante. Estos datos originalmente se encuentran guardados en *Microsoft Access*, pero han sido portados a CSV para ser fácilmente tratados. Las tablas usadas son:

4.1.1 Aircraft

Esta tabla contiene la información de cada avión de la flota. Sus columnas son las siguientes:

- ***Aircraft Type***: El fabricante produce diferentes tipos de aeronaves. Para poder distinguirlos se le asigna un nombre a cada tipo.
- ***Aircraft Serial Number***: Para distinguir internamente cada aeronave individualmente, el fabricante asigna un número de serie a cada aeronave producida. Esto permite seguirle el rastro independientemente del operador(dueño).
- ***Born Date***: Indica la fecha en la que la aeronave fue totalmente ensamblada.
- ***Death Date***: Indica la fecha en la que la aeronave fue retirada o estrellada. Se deja vacío si sigue activo.
- ***Country***: Región donde la aeronave ha sido registrada.

- ***Registration/Tail Number***: Código único identificador que la aeronave recibe de la National Avion Authority (NAA) de la región. Si se cambia de región, la otra NAA debe generar otro código.
- ***Customer***: Compañía propietaria de la aeronave en la actualidad.
- ***Historical Information***: Contiene datos relevantes sobre cada aeronave (inactividad, accidentes, etc).

Tras hacer una selección de parámetros finalmente nos quedamos con los siguientes datos:

ID	*Aircraft Type*	*Aircraft SN*	*Register*	*Born Date*	*ICAO24*

El *ID* es un parámetro propio de la base de datos para relacionarse con otras tablas. El *ICAO24* es un código similar a *Registration* que usado para obtener los datos de vuelo como se verá más adelante.

4.1.2 Aircraft Flight Hours

Esta tabla contiene todas las horas de vuelo registradas. Originalmente tiene estas columnas:

- ***Aircraft Type***: Equivalente a la anterior tabla.
- ***Aircraft SN***: Equivalente a la anterior tabla.
- ***Record Date***: Fecha cuando se reportaron las horas de vuelo acumuladas.
- ***Flight Hours***: Horas de vuelo acumuladas.
- ***Landings***: Número de aterrizajes. Muchas veces no tenemos este dato y hay que estimarlo.
- ***Source***: Origen de los datos.

Sustituimos *Aircraft Type* y *Aircraft SN* por el *ID* de la anterior tabla para de esta manera simplificar la forma de relacionar una aeronave con sus horas de vuelo.

En caso de no tener todas las horas de vuelo o no tener todos los aterrizajes, se rellenan utilizando una regresión lineal.

4.1.3 Defectos

Finalmente, los datos de defectos que contienen los detalles de los fallos reportados de la aeronave. Cada defecto consta de la siguiente información:

- ***Defect Date***: Fecha cuando el defecto sucede. Esto ayudará a marcar que vuelos tienen defectos.
- ***Aircraft Type***: Equivalente a la anterior tabla.

- ***Aircraft SN***: Equivalente a la anterior tabla.
- ***ATA Chapter***: Muestra información de qué sistema falló. Permite la clasificación de defectos por sistemas.
- ***Aircraft Flight Hours***: Horas de vuelo acumuladas en el momento de informar el fallo.
- ***Part Number***: *PN* del ítem que ha fallado.
- ***Part Serial Number***: *SN* del ítem que ha fallado.
- ***Part Operating Hours***: Horas acumuladas operativas hasta el fallo.
- ***Part Name***: Nombre del componente.
- ***Desciption***: Descripción del fallo detectado y las consecuencias ocasionadas.
- ***Damage Effect***: Después de recibir el *root cause analysis* del fallo, el *damage effect* debe ser definido. El defecto debe ser categorizado.
- ***Finding***: Otorgado por el proveedor. Explica la causa del defecto y las medidas a tomar para corregir el fallo.
- ***Operator***: Operador de la aeronave en el momento de fallo.

Una vez más sustituimos por el *ID*, filtramos los defectos por fechas entre Enero de 2016 y la actualidad y finalmente también filtramos por *ATA Chapter* eligiendo aquellos que tienen más relación con fallos producidos por el clima.

4.2 Flight Data

4.2.1 FlightRadar24

Flightradar24 [10] es un servicio de seguimiento de vuelos global que proporciona información en tiempo real sobre miles de aeronaves alrededor del mundo. También permite mirar datos históricos por aerolínea, aeronave, *aircraft type*, región o aeropuerto. Los datos obtenidos vienen de múltiples fuentes:

- ***Automatic Dependent Surveillance* (ADS-B)**: Usa un gran número de receptores terrestres que colectan datos de cualquier aeronave en su área local que tengan equipados un transmisor ADS-B y envía sus datos en tiempo real. Los transmisores usan el GPS y otros sensores para enviar datos del vuelo.
- ***Multilateration* (MLAT)**: Todos los tipos de aeronaves son visibles en las áreas cubiertas por MLAT incluso sin ADS-B, pero mientras que Europa tiene una cobertura del 99%, solo una pequeña parte de Estados Unidos queda cubierta. Se requieren al menos 4 receptores para calcular la posición de la aeronave.
- ***Satellite***: Satélites equipados con ADS-B recolectan datos fuera del rango terrestre.

- ***North America Radar Data***.
- ***FLARM***: Una versión simple de ADS-B con un rango más corto.
- ***Federal Aviation Administration* (FAA)**: En Estados Unidos, los datos recogidos por la autoridad local, pero con un cierto retraso.

4.2.1.1 Servicios

FlightRadar24 permite a sus usuarios acceder a los datos de vuelo de los últimos 7 días gratis. Para tener un mayor rango temporal es necesario un plan de pago, pero no es posible trabajar con él ya que las descargas paralelas/masivas son ilegales en dicha plataforma.

Para solucionar esto último, *FlightRadar24* ofrece un servicio de datos adicional a petición donde se paga según la dimensión de los datos a descargar. Los formatos disponibles son CSV o JSON. La información es la siguiente:

- ***Flight Number***.
- ***Registration***.
- ***Speed***.
- ***Heading***.
- ***Latitude***.
- ***Longitude***.
- ***Altitude***.
- ***Aircraft Type***.
- ***Callsign***.
- ***Origin***.
- ***Destination***.

Se encuentra repartida en dos tipos de ficheros separados que registran los datos cada 5 minutos:

- El primer tipo de fichero lista todos los vuelos individuales, separados por la fecha UTC.
- El segundo tipo de fichero contiene las posiciones de vuelo para cada vuelo dentro del primero.

```
20170901_flights.csv
flight_id,aircraft_id,reg,equip,callsign,flight,schd_from,schd_to,
real_to,reserved
246429460,4703687,LN-LND,B788,NAX7005,DY7005,ARN,JFK,JFK,

20170901_246429460.csv
snapshot_id,altitude,heading,latitude,longitude,radar_id,speed,squawk
1504224671,40000,226,53.90662,-58.48219,2560,439,0
1504224761,40000,226,53.78264,-58.70189,2560,441,24689
1504224822,39975,226,53.69587,-58.85457,2560,443,24689
1504224884,40000,226,53.60788,-59.00835,2560,442,24689
...
```

Figure 4.1: Fichero de ejemplo CSV de FlightRadar24

4.2.2 OpenSky Network

OpenSky Network [11] es una asociación sin ánimo de lucro ubicada en Suiza. Su objetivo es mejorar la seguridad, confiabilidad y eficiencia del uso del espacio aéreo dando acceso a una base de datos pública respecto al tráfico aéreo. Opensky consiste en múltiples sensores conectados a Internet por voluntarios, compañías que apoyan la iniciativa y organizaciones académicas/gubernamentales. Todos los datos se archivan en una enorme base de datos histórica. Estos datos se usan principalmente por investigadores de diferentes campos.

Las tecnologías principales detrás de *OpenSky* son el ADS-B y el Mode S. Estas registran datos en tiempo real sobre la frecuencia pública de 1090 MHz.

4.2.2.1 Servicios

En comparación a FlightRadar, *OpenSky* ofrece acceso gratuito a todos los datos mediante su API. Adicionalmente, se puede pedir acceso a un servicio premium para investigadores, académicos o autoridades del gobierno.

La comunidad de *OpenSky* [20] ha trabajado en herramientas [21] para facilitar la descarga de datos creando librerías basadas en *Python*, *R* y *Matlab*.

4.2.3 Decisión Final

FlightRadar24 es el servicio más conocido y fue la primera elección para obtener los datos de vuelo. Para probar el servicio se contrató una prueba del plan premium. Después de diversas descargas paralelas se hizo visible un aviso que advierte respecto los Términos del Servicio y su política de descargas no automatizadas.

Después de una larga búsqueda, se encontró *OpenSky*, más abierta y centrada en la investigación. Una vez creada la cuenta, se pidió acceso al servicio premium con éxito.

4.2.4 Datos de OpenSky

De todas las herramientas de *OpenSky*, *pyOpenSky* [25] era la que más se adecuaba al proyecto. Desarrollada por Junzi Sun, proporciona una interfaz *Python* a la base de datos histórica de *OpenSky* (*Impala Shell*). Esta librería permite:

- Filtrar y descargar datos ADS-B (*state vectors*).
- Filtrar y descargar mensajes Mode S en raw (*rollcall replies*).
- Decodificar los mensajes Mode S.

La librería tiene diferentes clases que ayudan a diferentes funcionalidades. La clase *OpenskyImpalaWrapper* permite descargar mensajes *raw* y datos ADS-B formando *state vectors*. Podemos definir distintos parámetros:

- **Tipo de datos**: Podemos seleccionar *raw* o ADS-B.
- **Fecha de inicio y final**: Indica el periodo de interés. Solo los *state vectors* entre estas fechas serán buscados.
- ***ICAO24***: Es posible filtrar los *state vectors* y obtener solo los relacionados con nuestra lista de aeronaves (la lista encontrada en FRACAS).

Los *state vectors* obtenidos son un *Pandas DataFrame* que contienen los siguientes campos:

- ***ICAO24*(*String*)**: El identificador de 24 bits que se usa para seguir aeronaves concretas sobre diferentes vuelos.
- ***Latitude***(*Float*): La coordenada latitud.
- ***Longitude***(*Float*): La coordenada longitud.
- ***Velocity***(*Float*): La velocidad de la aeronave respecto a la tierra en metros por segundo.
- ***Onground***(*Boolean*): Señal que indica si la aeronave se encuentra en el aire o en tierra.
- ***Baroaltitude***(*Float*): Altitud medida por el barómetro que depende de varios factores como el clima.
- ***Geoaltitude***(*Float*): Altitud medida usando el sensor GNSS (GPS).
- ***Heading***(*Float*): Dirección del movimiento representado como el ángulo en el sentido del reloj desde el norte geográfico.
- ***Vertical Speed***(*Float*): La velocidad vertical de la aeronave en metros por segundo. Un número negativo indica que la aeronave desciende y uno positiva un ascenso.

Out[3]:

	time	lat	lon	onground	baroaltitude	geoaltitude	velocity	heading	vertspeed
0	2016-06-11 21:36:30	39.104	-77.385	False	6088.38	6088.38	128.006	217.653	0.325
1	2016-06-11 21:36:31	39.104	-77.385	False	6088.38	6088.38	128.006	217.653	0.325
2	2016-06-11 21:36:32	39.100	-77.390	False	6096.00	6088.38	123.776	217.060	0.975
3	2016-06-11 21:36:33	39.100	-77.390	False	6096.00	6088.38	123.776	217.060	0.975
4	2016-06-11 21:36:34	39.100	-77.390	False	6096.00	6088.38	121.514	217.258	0.975
...	...	...	...	...	...	...	...	...	...
40209682	2020-11-30 19:45:11	44.455	-73.349	False	960.12	876.30	48.612	129.417	-2.276
40209683	2020-11-30 19:45:12	44.455	-73.349	False	960.12	876.30	48.612	129.417	-2.276
40209684	2020-11-30 19:45:13	44.455	-73.349	False	960.12	876.30	48.612	129.417	-2.276
40209685	2020-11-30 19:45:14	44.455	-73.349	False	960.12	876.30	48.612	129.417	-2.276
40209686	2020-11-30 19:45:15	44.455	-73.349	False	960.12	876.30	48.612	129.417	-2.276

40209687 rows × 9 columns

Figure 4.2: Ejemplo de DataFrame descargado de pyOpensky

El estudio al principio planteó descargar el máximo número de vuelos de aeronaves posibles. Después de descargar los primeros ejemplos quedó claro que había que cambiar el objetivo: cada aeronave cuenta con unos 50.000-100.000 *state vectors* (desde 2016), la descarga toma sobre unas 14h y ocupan aproximadamente 0.5-1 GigaByte (sin comprimir).

Por tanto, se cambió el planteamiento: seleccionamos las 9 aeronaves con mayor número de defectos y que cumplan los filtros anteriormente explicados. Después de los primeros test, se decidió descargar datos de 6 aeronaves más que el modelo no había visto nunca para verificar los resultados. Finalmente se descargaron 3 aeronaves más haciendo un total de 18.

Como se puede ver, esta fue una de las partes más duras del estudio, además el límite de descargas en paralelo y algunos fallos en el acceso a la API hacían que el proceso se ralentizara más.

4.3 Datos Meteorológicos

Contamos con los datos de vuelo y de fallo, pero todavía no tenemos datos meteorológicos con los que relacionar los anterior. Esta información nos indicará a qué condiciones fue expuesta cada aeronave en cada instante. Necesitamos los datos meteorológicos de cada posición de la ruta de cada vuelo durante la fecha que la aeronave sobrevoló la zona.

Varias fuentes de datos aparecieron, pero ninguna terminaba de cumplir con los requisitos históricos (una vez más limitaciones temporales y de descarga). Finalmente, la opción seleccionada fue una API gratuita con una librería de Python asociada llamada Meteostat.

4.3.1 Meteostat

Meteostat [16] proporciona una API simple para acceder a datos meteorológicos. Las observaciones históricas obtenidas de Meteostat provienen de interfaces públicas (normalmente gubernamentales). Entre estas destacamos *National Oceanic and Atmospheric Administration* (NOAA) y el servicio nacional de meteorología alemán (DWD).

Estos datos son tomados de las estaciones meteorológicas alrededor del mundo. Es posible descargar los datos raw de cada estación o incluso pedir una interpolación de los restantes.

Por cada posición y fecha se extraen las 50 estaciones más cercanas a la posición de la aeronave y entre esas 50 estaciones se busca el conjunto de 3 estaciones más cercano que triangulan la posición del avión. Para obtener los datos finales se realiza una suma ponderada de los datos de cada estación. Los pesos para la suma se consiguen calculando las coordenadas baricéntricas entre el triángulo y la posición de la aeronave

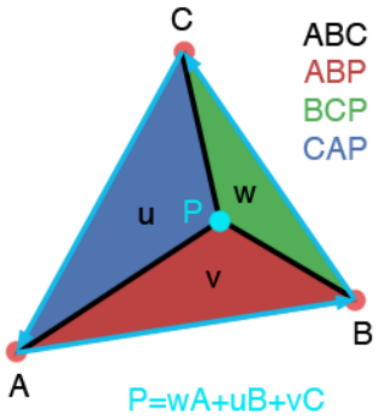

Figure 4.3: Obtención de las coordenadas baricéntricas [12].

donde $A, B, C, P \in R^2$ son puntos 2D que representan las posiciones de las estaciones y la aeronave y $u, v, w \in R$ son escalares que representan el peso de cada estación.

4.3.2 Estructura de los datos meteorológicos

Los datos descargados indican las condiciones de cada hora del día de una determinada estación. Tienen la siguiente forma:

- ***Station*** (*String*): El identificador de la estación meteorológica.
- ***Time*** (*Datetime*): Fecha de observación.
- ***Temperature*** (*Float*): Temperatura del aire en °C.
- ***Dewpoint*** (*Float*): Punto de rocío en °C.
- ***Relative Humidity*** (*Float*): Humedad relativa en porcentaje (%).
- ***Precipitation*** (*Float*): El total de precipitaciones de una hora en mm.
- ***Snow*** (*Float*): La profundidad de nieve en mm.
- ***Wind Direction*** (*Float*): La dirección promedia del viento en grados (°).
- ***Wind Speed*** (*Float*): La velocidad del viento promedia en km/h.
- ***Wind Peak Gust*** (*Float*): La ráfaga de viento máxima en km/h.

- ***Pression*** (*Float*): La presión del aire promedia a nivel del mar en hPaLa velocidad del viento promedia en km/h.
- ***Time Sunshine*** (*Float*): El tiempo total de sol en minutos durante una hora.
- ***Weather Condition Code*** (*Float*): El código de condición meteorológica que describe el clima.

	A	B	C	D	E	F	G	H	I	J	K
1	temp	dwpt	rhum	prcp	snow	wdir	wspd	wpgt	pres	tsun	coco
2	-2,6775	-13,3841	43,8654	0	0	285,9487	18,6889	0	1014,1395	0	0
3	-3,5256	-13,8943	45,051	0	0	291,7308	8,5688	0	1014,7382	0	0
4	-4,0235	-13,0038	50,7058	0	0	288,3525	18,0683	0	1015,5783	0	0
5	-4,4804	-14,7567	44,8904	0	0	304,0929	19,4406	0	1016,2598	0	0
6	-4,7285	-15,301	43,7048	0	0	272,5288	19,1004	0	1016,8078	0	0
7	-5,2847	-15,5925	44,4478	0	0	294,1346	17,3241	0	1017,174	0	0

Figure 4.4: Fichero de ejemplo CSV de Meteostat

Chapter 5

Procesamiento de Datos

Se han repasado las tres fuentes de datos, sus parámetros y estructura, pero falta ver cómo relacionamos estos datos para unificarlos en un único conjunto de datos.

5.1 Transformaciones

Se necesita juntar las tres fuentes de datos. Como se comentó en el capítulo anterior, los datos meteorológicos se obtuvieron de las rutas de vuelo realizadas por la aeronave. El primer paso es relacionar las condiciones meteorológicas con cada posición de los datos de vuelo (*time*, *latitude* and *longitude*) y realizar una interpolación de los valores faltantes (ya que solo tenemos las condiciones de las horas en punto). Después obtenemos los datos de FRACAS y relacionamos cada momento del vuelo con la información de la aeronave y sus defectos. Se utilizan las horas acumuladas de vuelo y una flag que indica si ese día se reportó un fallo. La siguiente figura muestra cómo se distribuyen todos los datos:

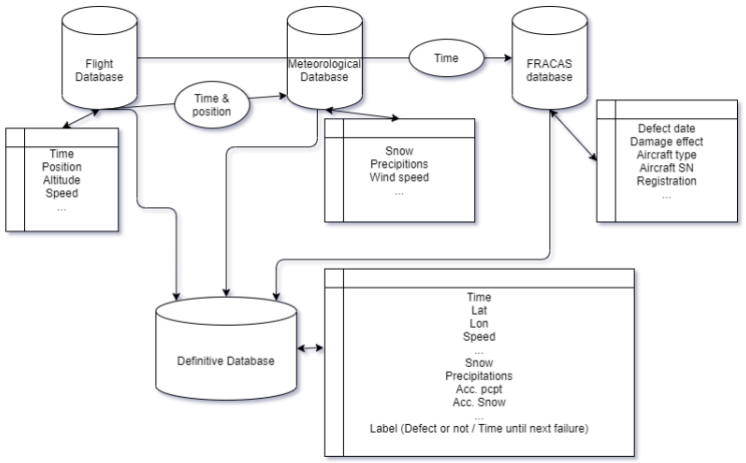

Figure 5.1: Unión de las tres fuentes de datos

Una vez tenemos los datos definitivos: redondeamos los decimales a la tercera cifra, eliminamos los *NaN* resultantes de las uniones. El resultado es un enorme volumen de datos debido a tener la posición de la aeronave cada segundo. Para reducir el volumen agrupamos los datos en minutos diferentes y calculamos la media de los valores (entre minutos si habrá cambios más significativos). Para finalizar transformamos todas las variables categóricas en valores numéricos y realizamos una normalización con tipificación Z.

5.2 Estructura Final

Una vez ya tenemos todos los datos unidos debemos dividir los datos en la matriz de características X y el valor a predecir y.

5.2.1 Características

La matriz de características contiene las variables independientes a procesar. Consta de todos los valores menos la característica a predecir. En nuestro estudio está formada por:

- ***ICAO24***.
- ***Latitude***.
- ***Longitude***.
- ***Onground***.
- ***Baroaltitude***.
- ***Geoaltitude***.
- ***Velocity***.
- ***Heading***.
- ***Vertical Speed***.
- ***Temperature***.
- ***Dewpoint***.
- ***Relative Humidity***.
- ***Precipitation***.
- ***Snow***.
- ***Wind Direction***.
- ***Wind Speed***.
- ***Wind Peak Gust***.
- ***Presion***.

- ***Time Sunshine***.
- ***Weather Condition Code***.
- ***Flying Hours***.
- ***Hour to defect/defect***.

Esta última variable como veremos adelante cambiará dependiendo el modelo que usemos.

5.2.2 Clase

Por otro lado, tenemos la clase que conforma el valor a predecir basándose en la matriz de características. Según el modelo que usemos tendremos defect como valor a predecir o hours to defect.

- ***Defect***: Usado para clasificación binaria. Toma como valores 0 (ausencia de fallo) y 1 (presencia de fallo).
- ***Hours to defect***: Usado para clasificación multiclase. Tenemos 10 clases diferentes que expresan diferentes intervalos de tiempo. La clase 0 representa los momentos donde hay fallo (0 horas para el siguiente fallo porque está sucediendo en ese mismo momento). A partir de la clase 1 cada intervalo es de 200h, es decir, la clase 1 comprende el intervalo (0,200], la clase 2 (200,400] y así hasta llegar a la clase 10 que comprende todos los valores mayores a los demás intervalos.

5.3 Desequilibrio de clases

Durante la etapa inicial del proyecto nuestros modelos mostraron un rendimiento pobre. Mirando bien los datos se pudo notar un gran desequilibrio de clases en el primer modelo y un pequeño desequilibrio en el segundo.

El desequilibrio de clases sucede cuando tenemos muchas muestras de una clase o varias clases concretas y muy pocos de las demás (es aún más exagerada en clasificaciones binarias).

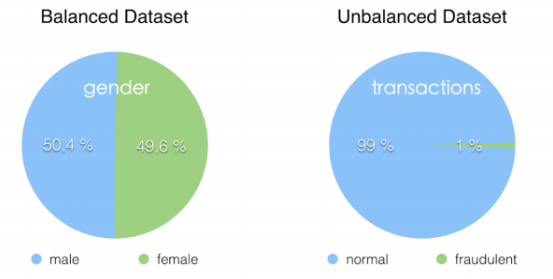

Figure 5.2: Ejemplo de un desequilibrio de clases [13].

Este problema puede ser solucionado mediante un resampling de nuestros datos (añadiendo más datos de las clases minoritarias, eliminando datos de las clases mayoritarias o ambas). Hablamos entonces de métodos de *oversampling* y *undersampling*.

Figure 5.3: Undersampling vs Oversampling [13].

En este proyecto se han usado diferentes técnicas de *oversampling*:

5.3.1 ADASYN-SMOTE

Tanto Synthetic Minority Oversampling Technique (SMOTE) como Adaptive Synthetic (ADASYN) usan el mismo algoritmo para usar nuevas muestras. Dada una muestra X_i , una nueva muestra X_{new} se puede generar considerando sus k-vecinos más cercanos.

$$X_{new} = X_i + \lambda \cdot (X_{zi} - X_i) \tag{5.1}$$

Donde λ es un número aleatorio entre 0 y 1. Esta interpolación creará una muestra en medio de la línea creada por X_{zi} y X_i.

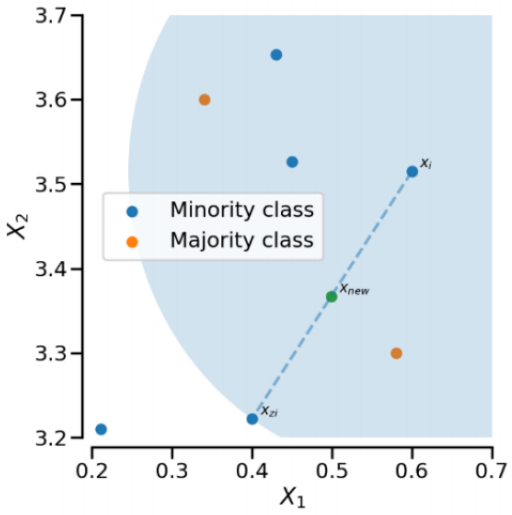

Figure 5.4: Funcionamiento de SMOTE [14].

ADASYN es una versión mejorada de SMOTE. Después de crear estas muestras añade un pequeño valor aleatorio a los puntos haciendo más realista la nueva muestra. En otras palabras, en vez de ser una nueva muestra totalmente correlacionada linealmente con sus muestras originales, tiene una pequeña variación, por ejemplo, un poco más disperso. Utilizamos ADASYN para corregir el desequilibrio en el modelo multiclase.

5.3.2 Borderline SMOTE

Otra variación del algoritmo original del SMOTE que funciona de clasificando cada muestra Xi en: Ruido (todos los vecinos cercanos son de una clase diferente a la de Xi).

- **Ruido**: Todos los vecinos cercanos son de una clase diferente a la de X_i.
- **En peligro**: Al menos la mitad de los vecinos cercanos son de la misma clase que X_i.
- **Seguro**: todos los vecinos cercanos son de la misma clase que X_i

De esta manera los datos pueden ser generados a partir de valores extremos los cuales pueden ser útiles para problemas como la detección de fraudes (o una clasificación binaria como la nuestra).

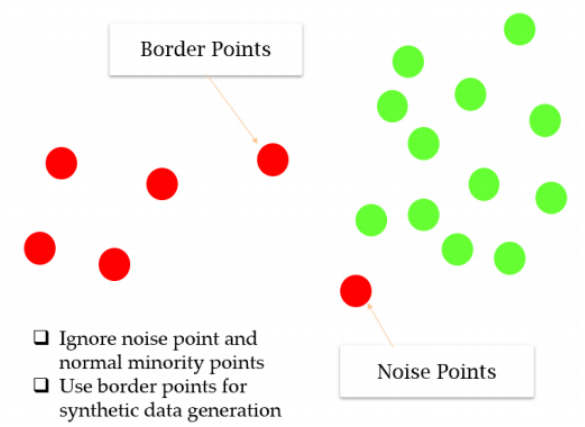

Figure 5.5: Funcionamiento de Borderline SMOTE

Este algoritmo fue utilizado para el modelo de clasificación binaria donde nos interesa generar más muestras de los defectos que son valores extremos.

Chapter 6

Implementación de los Modelos

En este capítulo se introducen los aspectos relacionados con la implementación de nuestro modelo, la evolución de estos y se muestran los resultados obtenidos. Adicionalmente se comparan los modelos obtenidos con otros algoritmos de *Machine Learning* más tradicionales.

6.1 Entorno de Desarollo

Los resultados presentados a continuación han sido obtenidos bajo el mismo sistema:

- AMD Ryzen 5 2600x @4.0GHz (6c/12th).
- NVIDIA RTX 3080 10GB VRAM @21Gbps.
- 32GB (2x16GB) DDR4 @3200MHz CL16 RAM.
- KINGSTON A400 240GB SATA SSD.

Y bajo el siguiente entorno:

- Ubuntu 20.04 LTS
- Python 3.6.10 - Anaconda, Inc - [GCN 7.3.0]
- Tensorflow 2.5.0
- Keras 2.5.0
- Scikit-learn (sklearn) 0.23.2
- Numpy 1.19.2
- Pandas 1.1.3
- Imblearn 0.8.0

6.1.1 Keras, Tensorflow y Scikit-learn

Keras es un framework de *Deep Learning* escrito en *Python* que proporciona una interfaz simple para crear y entrenar modelos de *Deep Learning*. Se ha seleccionado por la facilidad que tiene su API para el desarrollo. Proporciona todas las herramientas necesarias para prototipar la mayoría de modelos.

Tensorflow es una plataforma *open source* para *Machine Learning* creada por *Google*. Conceptualmente basada en los grafos de flujo de datos donde cada arista representa un tensor y cada nodo una operación en dicho tensor. *Keras* usa como *backend* esta plataforma.

Scikit-Learn es una librería *open source* para *Machine Learning*. Proporciona varias implementaciones de modelos conocidos, herramientas de procesamiento, selección y evaluación de datos entre otras cosas. Es una de las librerías más populares en *GitHub*.

6.2 Métricas

El módulo *sklearn.metrics* de *Scikit-learn* implementa una serie de funciones de emphLoss, score y utilidades para medir el rendimiento de nuestro modelo. En nuestro estudio hemos utilizado la función *precision_recall_fscore_support* que computa la *precision, recall, f-score* y *support* de nuestro modelo respecto la clasificación que ha realizado.

- ***Accuracy***: Mide el porcentaje de valores detectados correctamente respecto el total.

$$accuracy = \frac{VerdaderosPositivos + VerdaderosNegativos}{VerdaderosPositivos + VerdaderosNegativos + FalsosPositivos + FalsosNegativos} \tag{6.1}$$

- ***Precision***: Mide el porcentaje de positivos detectados que son verdaderamente correctos. Evalúa qué tan correcto es el modelo.

$$precision = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosPositivos} \tag{6.2}$$

- ***Recall***: Mide el porcentaje de positivos reales detectados respecto del total de positivos.

$$recall = \frac{VerdaderosPositivos}{VerdaderosPositivos + FalsosNegativos} \tag{6.3}$$

- ***F-score***: Es la media harmónica entre *precision* y *recall*. Busca un balance entre ambas métricas.

$$fscore = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{6.4}$$

Otra función de especial interés es confusion_matrix, ya que evalúa los resultados construyendo la matriz de confusión, una representación visual de como se ha realizado la clasificación.

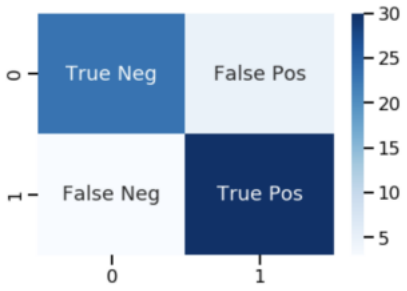

Figure 6.1: Ejemplo de Confusion Matrix

Finalmente, un valor muy interesante para este estudio es el *positive ratio* que mide el porcentaje de fallos reales que verdaderamente se han detectado (equivale al porcentaje de Verdaderos Positivos).

6.3 Clasificación Binaria vs Clasificación Multiclase

Durante el desarrollo del proyecto se planteó empezar por un modelo más simple que permita detectar si hay fallos o no asignando una *flag* binaria a predecir. Esto sirvió para ver qué tal se comportan los datos bajo un algoritmo de *Deep Learning* y poder ver que tan viable sería implementar un modelo más ambicioso.

Los resultados obtenidos animaron la creación de este segundo modelo que pretende estimar cuántas horas faltan hasta que se produzca el siguiente fallo.

6.3.1 Clasificación Binaria (Detección de Defectos)

6.3.1.1 Datos

La siguiente tabla muestra la cardinalidad de los datos originales:

Datos	**Filas**	**Columnas**
Datos de Entrenamiento + Validación	478193	22
Labels de Entrenamiento + Validación	478193	1
Datos de *Test*	445590	22
Labels de *Test*	445590	1

La distribución de las clases es la siguiente:

Datos	**Porcentaje 0s**	**Porcentaje 1s**
Entrenamiento + Validación	90.910%	9.090%
Test	96.937%	3.063%

Como hemos mencionado en capítulos anteriores, el problema del desequilibrio de clases genera nuevos datos sintéticos a partir de los originales. Después de aplicar *oversampling* la cardinalidad de nuestros datos cambia:

Datos	**Filas**	**Columnas**
Datos de Entrenamiento + Validación	869454	22
Labels de Entrenamiento + Validación	869454	1
Datos de *Test*	445590	22
Labels de *Test*	445590	1

Por tanto, la distribución de clases también se ve modificada:

Datos	**Porcentaje 0s**	**Porcentaje 1s**
Entrenamiento + Validación	50.000%	50.000%
Test	96.937%	3.063%

6.3.1.2 Estudio Preliminar

El primer modelo fue muy simple, consistía en unas pocas *fully-connected layers* con un pequeño número de neuronas.

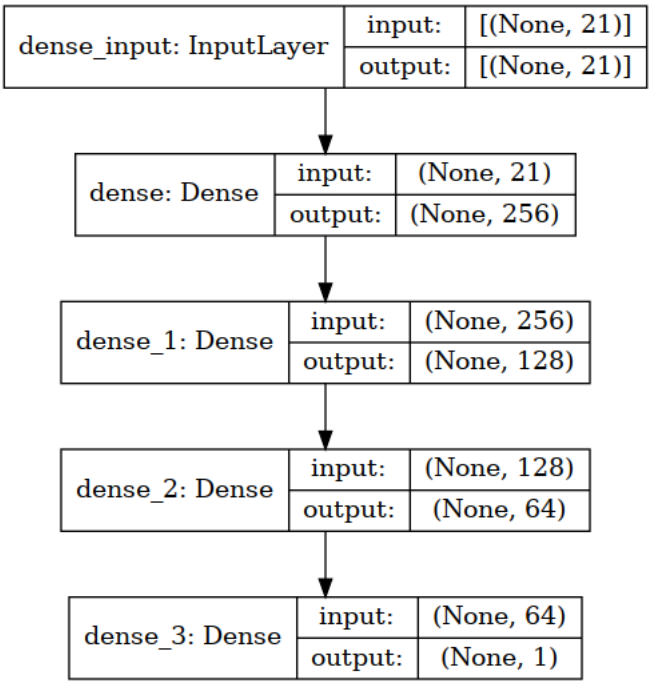

Figure 6.2: Primea Red Neuronal probada

A continuación, se verifica el comportamiento de nuestros datos y la influencia de distintos hiperparámetros en el rendimiento de nuestro modelo mediante un pequeño estudio preliminar.

Empezamos utilizando ADAM de optimizador, un *batch size* de 256, 100 *epochs* y MSE como función de *Loss*. A medida que avanzamos en el estudio, vamos incorporando los parámetros optimizados para la siguiente fase y así sucesivamente.

- **Optimizador**: El optimizador afecta directamente al rendimiento de nuestro modelo pues indica la forma en que los pesos se actualizarán y la manera de minimizar el error de la función de *Loss*. Vamos a comparar el rendimiento de ADAM, SGD, SGD + *momentum* + *nesterov* y RMSprop.

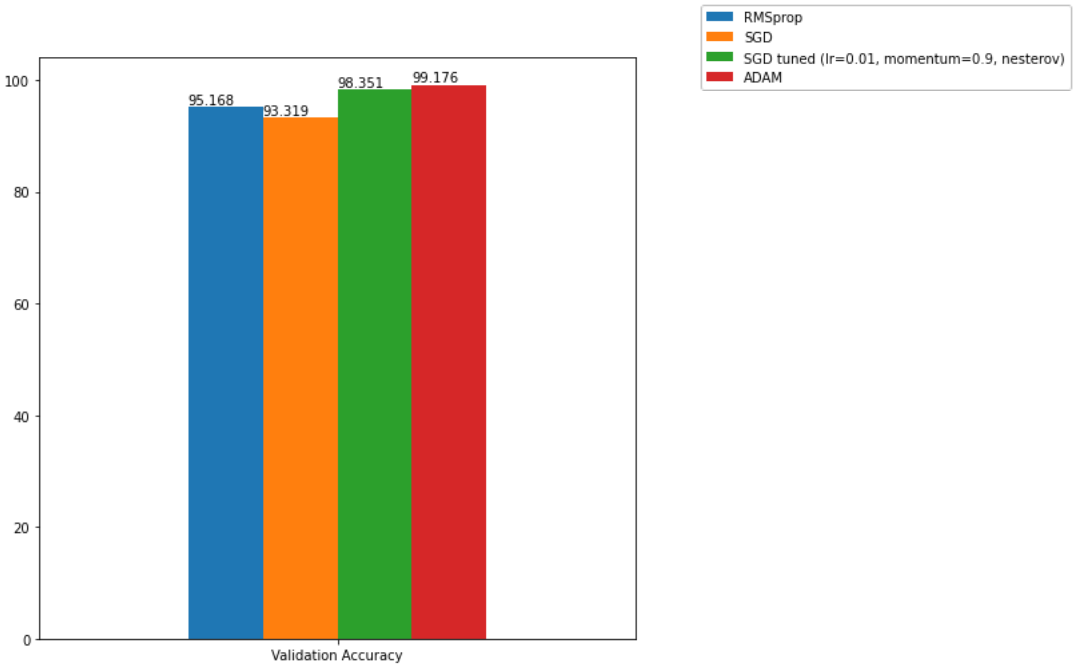

Figure 6.3: Comparación entre distintos optimizadores

Los resultados de la figura anterior muestran un rendimiento superior de ADAM (99.176%) respecto a sus rivales (98.351%, 95.168% y 93.319%) pero esto no es concluyente debido a la naturaleza aleatoria de estos experimentos. Lo que si podemos afirmar es que ADAM parece necesitar un menor número de épocas para minimizar la función (llega antes al mínimo) muy seguido del SGD + *momentum* + *nesterov*.

Teniendo esto en cuenta se ha decidido usar ADAM como el optimizador para nuestros modelos.

- ***Batch Size***: El *batch size* define el número de muestras que el modelo verá por iteración. Un *batch size* pequeño requiere menos memoria y hará que la mayoría de

las redes entrenen más rápido (más actualizaciones de parámetros). Por lo contrario, también hará que el gradiente calculado sea menos preciso.

Como se ha visto, el tamaño del *batch* puede afectar al rendimiento del modelo, por tanto, vamos a analizar el comportamiento de este con diferentes *batch size*.

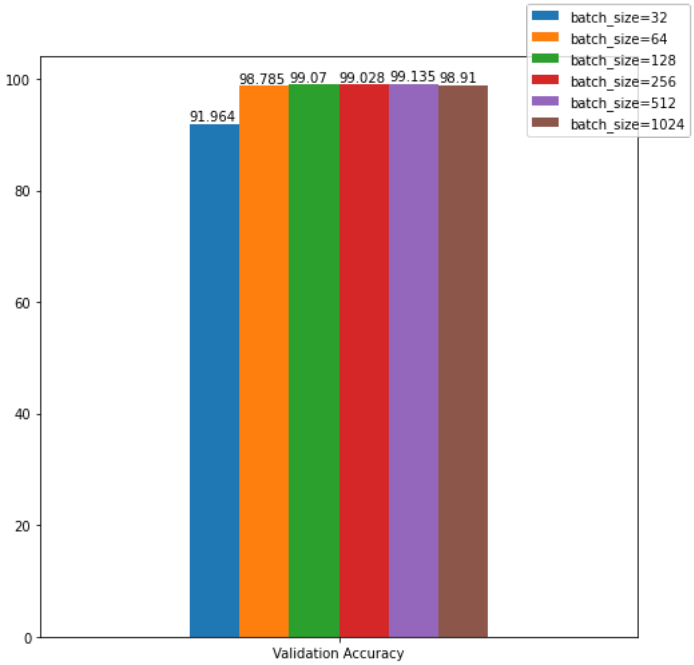

Figure 6.4: Comparación entre distintos Batch Sizes

Como se puede observar a excepción del *batch size* de 32 (91.964%), los demás rinden similarmente (≈99%). Esto puede ocurrir debido a que un *batch size* de 32 no permite calcular un gradiente tan preciso como los demás y no minimiza tan fácilmente la función.

Analizando los resultados se ve como cualquier *batch size* $\geq$ 64 podría ser válido y, por tanto, elegimos el más grande, 1024, para poder aprovechar cuanta más memoria posible de la GPU.

- ***Epochs***: Las *epochs* determinan el tiempo que el modelo estará entrenando. Un número de *epochs* muy bajo puede ser insuficiente para que el modelo alcance un rendimiento óptimo (por ejemplo, que no le de tiempo al optimizador a reducir la función de *Loss*. Por el contrario, un número de *epochs* muy grande puede llevar un gran *overfitting* para el modelo.

  Se han probado diferentes números de *epochs* para poder evaluar su impacto en el modelo y determinar qué valores son los óptimos.

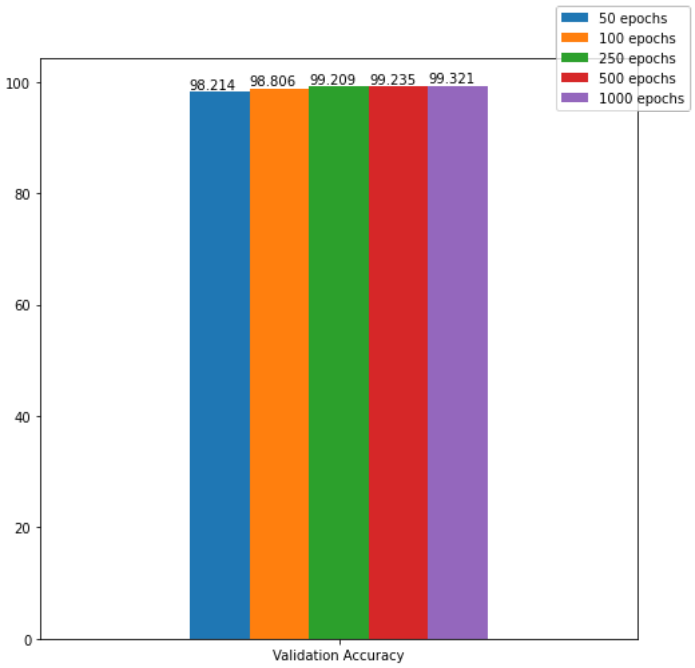

Figure 6.5: Comparación entre distintos números de epochs

La anterior figura muestra como todas las variaciones probadas tienen un rendimiento muy similar alrededor del 98-99% (siendo las diferencias mínimas).

Como todos rinden similar, se decide seleccionar el valor más pequeño ya que tiene menos posibilidad de presenciar *overfitting* y la relación rendimiento/tiempo de entrenamiento es la mejor.

- **Función de *Loss***: La función de *Loss* se utiliza para determinar el error entre los valores estimados por el modelo y el valor real. Al ser la función que usará el optimizador tiene una gran influencia sobre el rendimiento final del modelo.

  Por ello se han estudiado diferentes funciones de *Loss* que pueden ser útiles para nuestro problema.

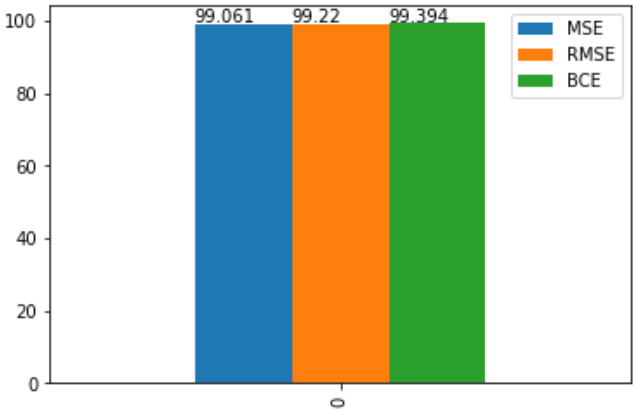

Figure 6.6: Comparación entre distintas funciones de Loss

Se puede ver como todas las funciones de *Loss* tienen un rendimiento muy similar, pero se decide optar por la función *Binary CrossEntropy* debido a que es una función más adecuada al contexto de nuestro problema de clasificación (expresa el error entre dos distribuciones probabilísticas).

Finalmente, el rendimiento obtenido por este primer modelo preliminar era muy bueno (alrededor de 97-98% de *accuracy*, *precision*, *recall* y *fscore* tanto para el conjunto de entrenamiento como para el de *test*).

Unos resultados tan buenos en el primer intento son impensables así que en el siguiente apartado se hablan de los errores encontrados y su solución.

6.3.1.3 Evolución

Los conjuntos de entrenamiento y *test* se habían creado a partir de un único conjunto y de hacer *splits* aleatorios. De esta manera los datos de *test* contenían trozos de vuelos que pertenecían al conjunto de entrenamiento y por tanto se obtenía este rendimiento.

Separando el conjunto de *test* de manera que esté formado por datos que la red nunca ha visto se observó como el rendimiento ahora era muy bajo (clasificaba todo como 0 y de los 1 solo predecía correctamente el <1%). Estábamos ante un problema de desequilibrio de clases. La siguiente tabla muestra los resultados obtenidos con este desequilibrio:

Modelo	Accuracy	Precision	Recall	F-score	Positive Ratio
Nuestro Modelo	98.462%	98.597%	99.861%	0.99225	0.278%
Logistic Regression	98.595%	98.595%	100.00%	0.99293	0.000%
Gaussian Naive Bayes	98.534%	98.594%	99.938%	0.99262	0.000%
KNeighbors	95.845%	98.601%	97.165%	0.97878	3.247%
Decision Tree	97.220%	98.576%	98.606%	0.98591	0.000%
Random Forest	98.595%	98.595%	100.00%	0.99293	0.000%
Support Vector Machine	98.595%	98.595%	100.00%	0.99293	0.000%

Después de probar diversos métodos de *oversampling* llegamos a la conclusión de que Borderline-SMOTE era el más adecuado (no solo por mostrar un mayor rendimiento sino por su propia definición). Con este cambio el modelo pasó a tener una peor *accuracy* pero incrementó el *positive ratio* correctamente en un 24-26%.

Para finalizar se modificó la red inspirándose en los modelos del estado del arte, obteniendo así una red CNN + LSTM.

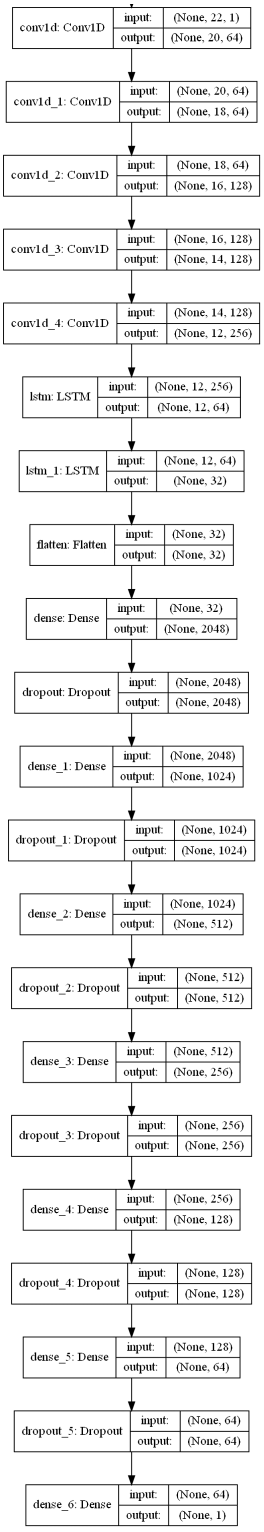

Figure 6.7: CNN + LSTM

6.3.1.4 Resultados Finales

Los resultados presentados se han llevado a cabo con los siguientes parámetros:

- ***Epochs***: 10
- ***Batch Size***: 1024
- ***Función de Loss***: *Binary CrossEntropy*
- **Optimizador**: ADAM

Estos cambios en la red sumados a tener unos datos más balanceados llevaron al modelo a unos resultados satisfactorios y realistas.

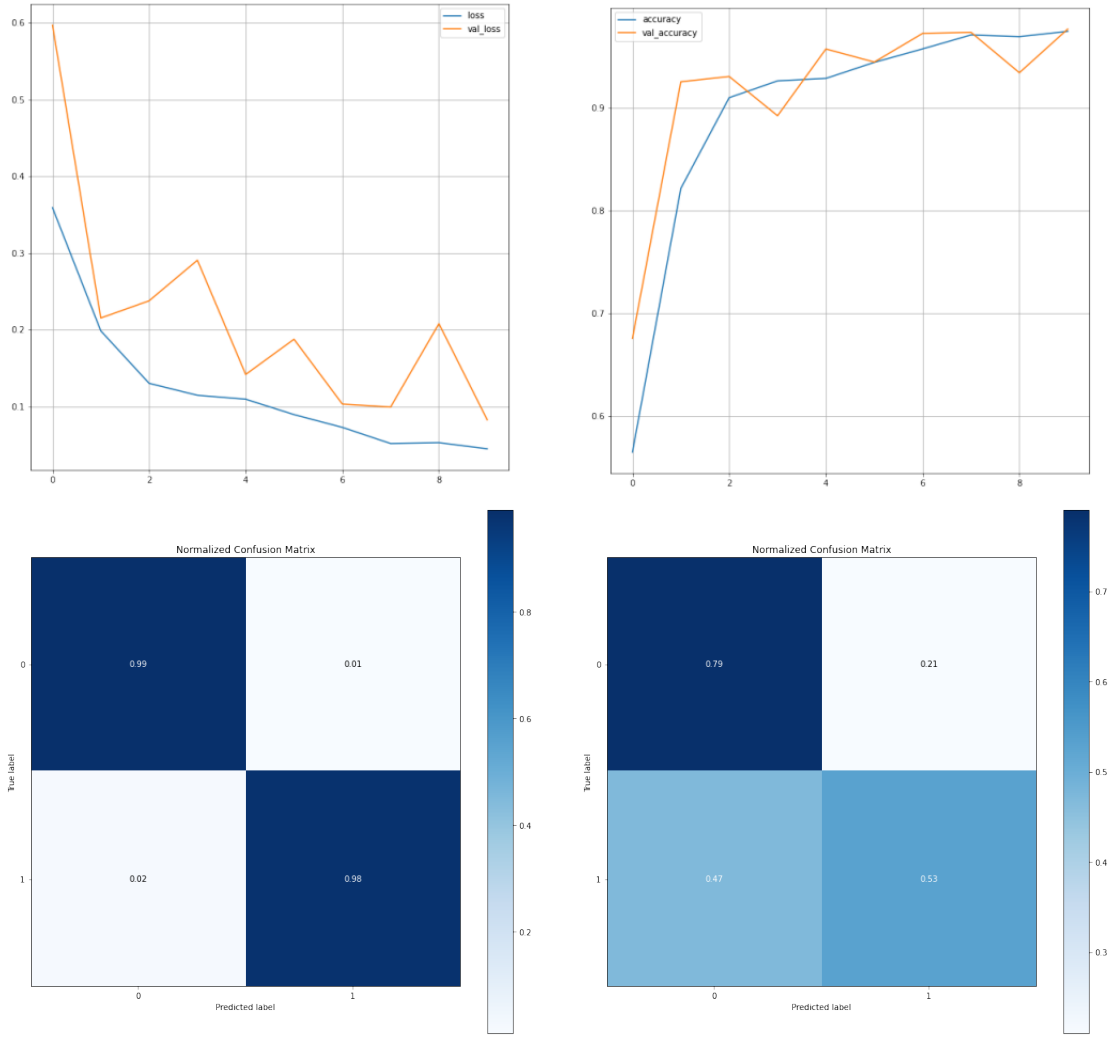

Figure 6.8: Resultados: Loss, Accuracy y Matriz de Confusión

El conjunto de validación tiene una accuracy del 98.093% con un *positive rate* del 98% mientras que el conjunto de *test* tiene una *accuracy* del 78.443% con un *positive ratio* alrededor del 55.135%.

6.3.1.5 Comparación con otros modelos

Finalmente, para concluir el estudio del primer modelo, hacemos una comparación con otros modelos clásicos de *Machine Learning*:

Modelo	Accuracy	Precision	Recall	F-score	Positive Ratio
Nuestro Modelo	78.444%	98.179%	79.232%	0.87694	55.136%
Logistic Regression	88.252%	97.215%	90.473%	0.93723	21.813%
Gaussian Naive Bayes	86.230%	97.238%	88.304%	0.92545	22.423%
KNeighbors	71.766%	97.441%	72.785%	0.83328	41.926%
Decision Tree	88.468%	97.245%	90.672%	0.93844	22.238%
Random Forest	67.699%	97.430%	68.485%	0.80433	44.827%
Support Vector Machine	86.829%	96.999%	89.172%	0.92921	15.416%

Notamos como nuestro modelo consigue la mayor precisión y *positive ratio* que son las métricas que más nos interesan puesto que queremos detectar correctamente los fallos. *KNeighbors* y *Random Forest* quedan con un rendimiento cercano a nuestro modelo comprobando así que se puede aprender a detectar fallos con nuestros datos y *machine learning*.

6.3.2 Clasificación Multiclase (Predicción de Intervalo de Horas)

6.3.2.1 Datos

La siguiente tabla muestra la cardinalidad de los datos originales (notamos que solo cambia el número de las *labels*):

Datos	Filas	Columnas
Datos de Entrenamiento + Validación	478193	22
Labels de Entrenamiento + Validación	478193	11
Datos de *Test*	445590	22
Labels de *Test*	445590	11

Una vez más la cardinalidad cambia después de aplicar *oversampling*. En este caso aumenta drásticamente debido a un mayor número de clases que balancear:

Datos	Filas	Columnas
Datos de Entrenamiento + Validación	16424092	22
Labels de Entrenamiento + Validación	16424092	11
Datos de *Test*	445590	22
Labels de *Test*	445590	11

6.3.2.2 Evolución

El modelo inicialmente se planteó como una regresión para crear la aproximación más cercana a la realidad, pero debido al pobre rendimiento de esta aproximación en todos los intentos nació la idea de trabajar con intervalos de tiempo.

Esta aproximación plantea transformar las horas hasta el defecto en *labels* que representan intervalos de tiempo y entonces realizar una clasificación para determinar en qué intervalo nos encontramos. Este planteamiento es suficientemente bueno ya que encontrándonos en un intervalo de tiempo muy cercano a defecto podemos generar una alerta.El diseño planteado por la red fue muy similar al del primer modelo, una CNN+LSTM:

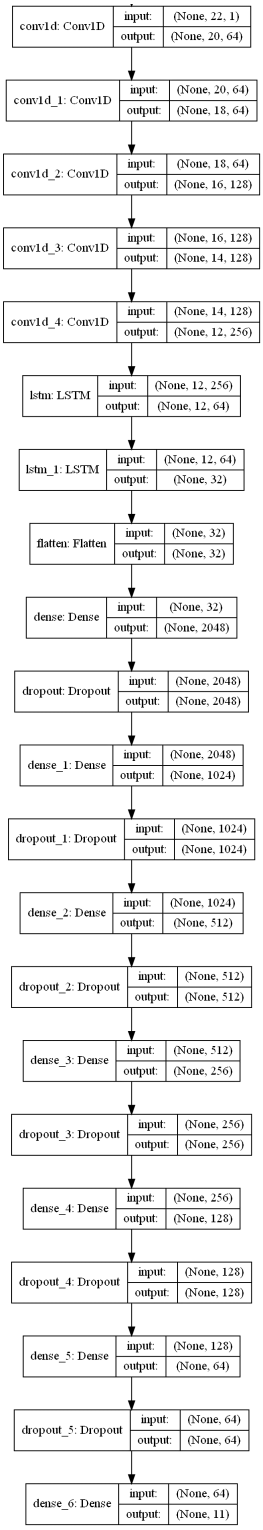

Figure 6.9: CNN + LSTM (intervalos)

Al principio los intervalos se generaron cada 100 h hasta las 5000 h. Esto llevó a una accuracy de 12.15%, un mal rendimiento. Otra vez hubo un desequilibrio de datos.

Esta vez se usó ADASYN en vez de Borderline-SMOTE para solucionar el problema (ya que la naturaleza del problema cambió y ADASYN era una mejora del propio SMOTE básico). Tras aplicar *oversampling* el resultado mejora dando entre un 22-24%, una mejora significativa pero no suficiente.

Se llegó a la conclusión que con abarcar 2000 h en intervalos de 200 h es suficiente para este problema. Esto redujo el número de clases a 10, dando más muestras por clase y haciendo más fácil clasificación (menos posibilidades entre las que elegir).

6.3.2.3 Resultados Finales

Los resultados presentados se han llevado a cabo con los siguientes parámetros:

- ***Epochs***: 10
- ***Batch Size***: 1024
- ***Función de Loss***: *Categorical CrossEntropy*
- **Optimizador**: ADAM

Los cambios realizados permitieron tener un rendimiento final cercano al del primer modelo (pese a ser un problema más complejo).

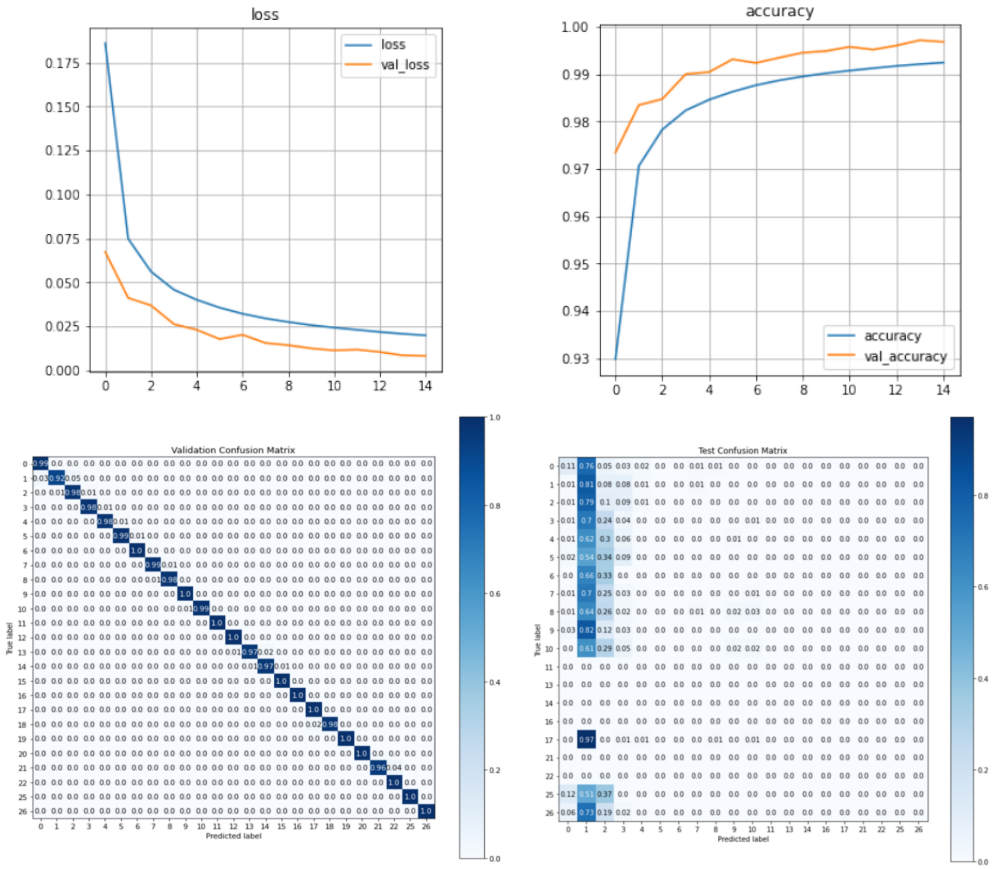

Figure 6.10: Resultados: Loss, Accuracy y Matriz de Confusión

Contrastando con la Confusion Matrix con la distribución de clases de *test* notamos como se han detectado correctamente mayoritariamente las horas hasta el fallo correspondientes a la clase 1, esto es verdaderamente importante ya que es el intervalo más cercano al error.

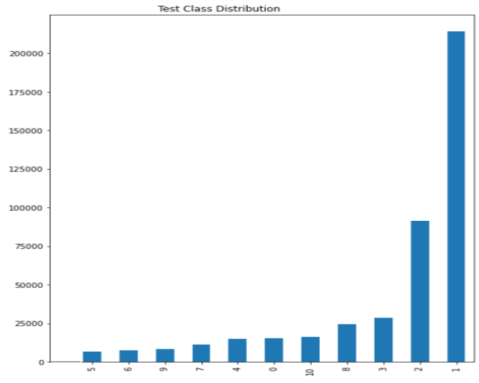

Figure 6.11: Distribución de clases del conjunto de test

El conjunto de validación llega al 99.3% de accuracy mientras que el conjunto de *test* llega al 46.4% de accuracy. En este caso la accuracy es mejor indicador de lo que lo era en la clasificación binaria ya que el clasificador no acierta todo por poner 0s (hay muchas más opciones que antes)

6.3.2.4 Comparación con otros Modelos

Como con el modelo anterior, vamos a evaluar nuestra red con algunos de los clasificadores más comunes.

Modelo	**Accuracy**
Nuestro Modelo	46.400%
Logistic Regression	7.897%
Gaussian Naive Bayes	1.620%
KNeighbors	5.192%
Decision Tree	33.915%
Random Forest	35.715%
Support Vector Machine	6.356%

La tabla muestra como nuestro modelo es el que tiene mejor rendimiento sobre el conjunto de *test*. *Decision Tree* y *Random Forest* son lo más cercanos con 33.915% y 35.715% de *accuracy* respectivamente. Los demás no llegan ni al 10% *accuracy*.

Chapter 7

Conclusiones y Planes para el futuro

En este capítulo vamos a tratar de sintetizar todos los resultados obtenidos en el proyecto para poder extraer unas conclusiones.

Como se ha explicado en los capítulos 1 y 3, un buen mantenimiento es crucial para extender la vida de una aeronave y evitar daños posteriores debidos al mal estado de algún sistema. Aunque históricamente se ha tratado de mejorar este aspecto implementando un sistema de Mantenimiento Preventivo, este ya está mostrando sus debilidades. En este estudio se plantea una aproximación al Mantenimiento Predictivo que plantea mejorar sus predecesores realizando tareas de mantenimiento solo cuando de verdad sea requerido.

Para ello hemos realizado una serie de experimentos tratando de analizar la detección de fallos relacionados con causas externas (principalmente causas meteorológicas) y comprobar la existencia de esta relación. Basándonos en los resultados obtenidos por nuestro modelo podemos concluir que el uso de arquitecturas basadas en redes neuronales recurrentes y convolucionales sobre datos basados en series temporales funcionan muy bien a la hora de detectar fallos en la aeronave y estimando el tiempo del fallo más cercano.

Nuestras mejores modelos han sido capaces de detectar entre un 45-55% de los casos anteriormente mencionados correctamente. Esta cifra debe contrastarse con el hecho de que no podemos asegurar que todos los fallos dentro de nuestro conjunto de datos tengan un origen relacionado con los factores climatológicos estudiados. Por tanto, se ha cumplido el objetivo: demostrar el potencial de predicción de estos datos aplicando *Deep Learning*.

De hecho, otro objetivo secundario del proyecto era demostrar este potencial para poder mostrar el estudio a diversas compañías con la premisa de que el acceso a unos mejores datos puede mejorar notablemente el modelo e incluso hacerlo polivalente (incluyendo más tipos de errores). Los datos de estas compañías provienen del sistema HUMS montado en todas las aeronaves (como hemos mencionado en capítulos anteriores).

Pese al buen rendimiento de nuestro modelo con nuestro conjunto de datos, es importante entender las limitaciones del mismo:

- Solo podemos clasificar fallos climatológicos.
- El modelo no contempla todas las condiciones a las que una aeronave puede verse sometida (por ejemplo, un fallo debido a unas condiciones climáticas nunca antes vistas por el modelo).
- Necesidad de unos datos más balanceados.

Por ello también creemos que posibles mejoras podrían alcanzarse si:

- Aumentamos la cantidad y la calidad de los datos (Inclusión de datos internos).
- Buscamos maneras alternativas de representar los datos.
- Probamos redes más profundas y con diferente estructura.
- Transformamos la clasificación de intervalos a un problema de regresión con los nuevos datos.

Estas mejoras abren la posibilidad de un mayor proyecto: el desarrollo de un módulo basado en Mantenimiento Predictivo para la herramienta *Robin RAMS* de *DMD Solutions*.

Bibliography

[1] Aircraft Maintenance. https://en.wikipedia.org/wiki/Aircraft_maintenance. Accedido en Enero de 2021.

[2] Mean Time Between Failures. https://en.wikipedia.org/wiki/Mean_time_between_failures. Accedido en Noviembre de 2020.

[3] Robin RAMS. https://robinrams.com. Accedido en Septiembre de 2020.

[4] Mobley R K. *Introduction to Predictive Maintenance.* 2002.

[5] Skywise from Airbus. https://skywise.airbus.com. Accedido en Abril de 2021.

[6] Yi-ting Lin. *Customer Churn Analysis and Prediction Modelling at Icecat , a Content Syndicator.* 2020.

[7] Esperanza García Gonzalo, Zulima Fernandez Múniz, Paulino Jose Garcia Nieto, Antonio Sanchez, and Marta Menéndez *Hard-rock stability analysis for span design in entry-type excavations with learning classifiers.* Materiales 9:531, 06 2016.

[8] Ruttala Sailusha, V. Gnaneswar, R. Ramesh, and G. Ramakoteswara Rao. *Credit card fraud detection using machine learning. In 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS).* páginas 1264–1270, 2020.

[9] Shruti Jadon. Shruti Jadon Medium. https://medium.com/@shrutijadon10104776/survey-on-activation-functions-for-deeplearning-9689331ba092. Accedido en Septiembre de 2020.

[10] *FlightRadar24.* https://www.flightradar24.com/about. Accedido en Diciembre de 2020.

[11] *OpenSky Network.* https://opensky-network.org/about/about-us. Accedido en Diciembre de 2020.

[12] *Ray Tracing: Rendering a Triangle.* https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates. Accedido en Marzo en 2021.

[13] German Lahera. *Unbalanced Datasets What To Do About Them.* https://medium.com/strands-tech-corner/unbalanced-datasets-what-to-do-144e0552d9cd. Accedido en Marzo de 2021.

[14] German Lahera. *Unbalanced Datasets What To Do About Them.* https://imbalanced-learn.org/stable/over_sampling.html. Accedido en Marzo de 2021.

[15] Dennis T. *Confusion Matrix Visualization* https://medium.com/@dtuk81/confusion-matrix-visualization-fc31e3f30fea. Accedido en Febrero de 2021.

[16] Meteostat API. https://dev.meteostat.net. Accedido en Enero de 2021.

[17] Z. Vintr and M. Vintr. *Tools for components reliability prediction. Safety and Reliability - Theory and Applications - Proceedings of the 27th European Safety and Reliability Conference*, ESREL 2017, (February):2187–2194, 2017.

[18] FIDES. *FIDES guide 2009 Edition A September 2010 Reliability Methodology for Electronic Systems.* French armament industry supervision agency FIDES, (September), 2010.

[19] Quim Martínez. *Reliability prediction algorithms for aircraft batteries. Bachelor's final project*, Universitat Politecnica de Catalunya, 2020.

[20] Junzi Sun and Jacco M. Hoekstra. *Integrating pymodes and opensky historical database*. EPiC Series in Computing, 67(December): 63–72, 2019.

[21] OpenSky Datatools. https://opensky-network.org/data/data-tools. Accedido en Enero de 2021.

[22] B. S. Dhillon *Reliability engineering*, 2016.

[23] Joel Levitt *Complete Guide to Preventive and Predictive Maintenance*, Second Edition. 1952.

[24] Panagiotis Korvesis. *Machine Learning for Predictive Maintenance in Aviation*, Panagiotis Korvesis To cite this version : HAL Id : tel-02003508 Thèse de doctorat de l' Université Paris-Saclay préparée a l'école Polytechnique par. M . Panagiotis Korvesis le Domaine de l ' Avia. 2019.

[25] Junzi Sun. *PyOpenSky. Python interface for OpenSky historical database.* https://github.com/junzis/pyopensky. Accedido en Enero de 2021

[26] Goodfellow-et-al-2016, *Ian Goodfellow and Yoshua Bengio and Aaron Courville*, MIT Press, http://www.deeplearningbook.org, 2016