UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL SANTA FE

PROYECTO FINAL DE CARRERA

“Prototipo de software basado en aprendizaje profundo para mantenimiento predictivo en la Industria 4.0”

ALUMNOS

Mauro José Pacchiotti - L.U. 13550 - email mpacchiotti@frsf.utn.edu.ar
Pablo Andrés Paletto - L.U. 19470 - email ppaletto@frsf.utn.edu.ar

DIRECTOR

Dr. Mariano Rubiolo

CODIRECTOR

Dr. Jorge Roa

AÑO

2021

Resumen

Este Proyecto Final de Carrera se propone estudiar la aplicación de distintos modelos de aprendizaje profundo para mantenimiento predictivo en la industria. En el marco del trabajo industrial con distintos tipos de equipamientos de diversas complejidades insertos en diferentes procesos industriales, el mantenimiento eficiente de estos equipos es una actividad fundamental para un desarrollo industrial sustentable.

En primera instancia se analizan distintos trabajos que estudian el tema del proyecto para conocer las distintas líneas de investigación y los avances logrados por estos trabajos. También se buscan y analizan distintos conjuntos de datos que puedan ser aplicados al proyecto y se estudian los tipos de modelos de aprendizaje profundo utilizados en el dominio así como sus características y posibilidades frente a los diferentes tipos de estructuras de datos que se utilizan.

Con los resultados de las primeras etapas se avanza en la selección de un conjunto de datos y un modelo, y la adecuación de estos. Con un modelo entrenado podemos comenzar el diseño y desarrollo del prototipo herramienta de software y realizar las pruebas de funcionamiento en tiempo real, simulando con series sintéticas la entrega de datos desde un equipo en la industria.

Finalmente, luego de haber obtenido de la industria un conjunto de datos real, se selecciona y entrena un modelo para este, se adecua el prototipo para la estructura de datos y por último se realizan las pruebas simulando en tiempo real la entrega de datos y obteniendo del predictor el estado futuro del equipamiento.

Índice

Índice de figuras

Índice de tablas

1 . Introducción

Desde los primeros talleres donde el hombre en grupos y con la utilización de maquinarias, comenzó a producir bienes, ocurrieron fallas en los equipamientos que necesitaron de la reparación de los mismos para poder continuar con la producción. Inicialmente los mismos operarios que trabajaban en los distintos procesos eran los que reparaban sus propias máquinas y herramientas cuando estas presentaban averías. Debido a que los trabajadores desarrollaban múltiples oficios, el elaborar un producto terminado para ofrecerlo en el mercado implicaba un alto costo en tiempo y dinero.

Buscando reducir la complejidad de las tareas y favorecer la división del trabajo para mejorar la productividad, las empresas comenzaron a separar las actividades de los trabajadores en dos grupos, actividades de producción y actividades de mantenimiento. En 1930 Henry Ford incorpora en su empresa el concepto de mantenimiento, inclusive, creando el puesto de Director de Mantenimiento para quien estaba a cargo de gestionar estas tareas.

Con la llegada de la segunda guerra mundial, las industrias debieron incrementar sus producciones para atender las demandas que imponía el mercado, esto llevó a un incremento de la jornada laboral, incluso casos en que la producción no se detenía, repercutiendo esto en el estado y en un mayor deterioro de los equipamientos industriales.

Las paradas de producción por fallas de las máquinas comenzaron a producir grandes pérdidas por lo que las empresas comenzaron a dar más importancia al mantenimiento y la aplicación de éste sobre los equipos. De esta manera, se comienza a realizar tareas de prevención en el mantenimiento para reemplazar las partes que sufren desgaste en los equipamientos antes que estos fallen y de esta forma reducir los tiempos de parada de producción.

La buena aplicación de mantenimiento en la industria tiene muchas ventajas: mejora la calidad de los productos, reduce los tiempos de producción, disminuye los riesgos de accidente de trabajo, reduce el tiempo perdido en paradas imprevistas que implican pérdida de materiales, aminora los fallos irreparables en los equipamientos y mejora la previsión de los tiempos de producción favoreciendo la elaboración de presupuestos, entre otras. La falta de un buen mantenimiento en la industria puede causar pérdidas como la elaboración de productos defectuosos, la pérdida de materia prima, accidentes laborales, paradas inesperadas

del proceso de producción, incumplimiento de los plazos de entrega con los clientes y averías irreparables en los equipos, entre las más importantes.[1]

El mantenimiento preventivo, también llamado programado, puede dar muy buenos resultados, sobre todo si está bien diseñado y se cuenta con suficientes fuentes de información experta para su implementación. Sin embargo, teniendo en cuenta que los proveedores de equipamientos industriales hacen hincapié en la confiabilidad de sus equipos, esto muchas veces impacta generando altos costos de mantenimiento.

La probabilidad de falla de componentes en los equipamientos es alta al principio y al final de su vida útil, debido a fallas en la instalación o en la fabricación del repuesto y al desgaste sufrido por la utilización del mismo. Por lo tanto las tareas de mantenimiento innecesarias aumentan la tasa de fallas cuando un elemento defectuoso se instala o cuando ocurre un error humano. Además, las estrategias de mantenimiento programado no tienen en cuenta contextos operativos, como número de arranques o parámetros ambientales entre otros, y estos afectan fuertemente la vida útil de los componentes.

Finalmente, el mantenimiento preventivo se basa erróneamente en la idea de que la probabilidad de ocurrencia de las fallas operativas aumentan exponencialmente en un momento determinado, entonces los componentes se reemplazan o reparan antes de que ese momento ocurra. Esta suposición no es cierta en muchos casos; hay varios patrones en los que la probabilidad de falla nunca aumenta, en estos casos, la probabilidad de falla es constante en el tiempo. Entonces un componente podría fallar en cualquier momento. Ejemplos de este fenómeno son los patrones de falla de componentes eléctricos y electrónicos, donde las tareas de sustitución en períodos de tiempo planificados no implican una prueba de fiabilidad. [2]

La Industria 4.0**,** también llamada cuarta revolución industrial, es un concepto utilizado en los últimos años para describir a la industria funcionando con la colaboración de distintas nuevas tecnologías que digitalizan, interconectan y optimizan procesos de manera nunca antes vista. Con la llegada de Internet de las Cosas (IoT) y las comunicaciones 5G se facilita la interconexión de equipamientos sin importar la distancia entre ellos, el almacenamiento en la nube permite el resguardo y alta disponibilidad de grandes cantidades datos.

Estas grandes cantidades de datos que empiezan a estar disponibles en las plantas industriales motiva la adopción de técnicas de Machine Learning para, mediante el análisis de estos datos, abordar requisitos y necesidades industriales. Para este trabajo en particular, el foco principal se pone en la predicción, es decir la capacidad de estimar y anticipar eventos sobre los activos industriales o predecir el estado del activo en un tiempo estimado.

En las próximas secciones se detallan los conceptos teóricos necesarios para entender los diferentes modelos, técnicas y artefactos empleados, como así también las metodologías y herramientas utilizadas en las diferentes tareas. Así mismo, se detalla el desarrollo del proyecto, finalizando con las conclusiones y los trabajos futuros.

1.1. Objetivos

1.1.1 Objetivos Generales

Este proyecto persigue como principal objetivo el desarrollo de un prototipo de herramienta de software para mantenimiento predictivo que, a partir de series de tiempo capturadas por diferentes tipos de sensores en un determinado equipo, pueda predecir el estado de éste o alguno de sus componentes en un futuro cercano, utilizando para ello modelos de aprendizaje profundo.

1.1.2 Objetivos Específicos

- Analizar la utilización de los modelos de aprendizaje profundo para técnicas de mantenimiento predictivo en la Industria.

- Investigar distintos conjuntos de datos de series temporales provistos por sensores en equipos industriales.

- Desarrollar un modelo de aprendizaje profundo que prediga el estado de un componente o equipo industrial a partir de los datos provistos por sensores de diferentes tipos.

- Desarrollar un prototipo de herramienta de software, que a partir del modelo de aprendizaje profundo obtenido, permita simular la predicción del estado del dispositivo sensado en tiempo real.

- Validar el modelo de aprendizaje obtenido con los diferentes conjuntos de datos estudiados y analizar los resultados.

1.2. Temporalidad

El desarrollo de este proyecto transcurrió en el período comprendido entre el mes de Noviembre de 2020 y el mes de Febrero de 2021.

2 . Marco Teórico

2.1. Inteligencia Artificial

La Inteligencia Artificial (I.A.) es una disciplina que data de mucho tiempo atrás teniendo como precursor de esta a Alan Turing. Según John McCarthy (quien acuñó el término Inteligencia Artificial en la conferencia de Dartmouth en 1956), la I.A. es la ciencia e ingeniería de hacer máquinas inteligentes, sin embargo no hay una definición unificada de que es la I.A. y distintos autores han desarrollado distintas. Algunas de ellas son:

- *"La inteligencia artificial es el estudio de cómo hacer que las computadoras hagan cosas que por el momento los humanos hacen mejor"* [3]

- *"... la ciencia de hacer que las máquinas hagan cosas que requerirían inteligencia si fueran hechas por el hombre"* [4]

- *"... la habilidad de una computadora digital o un robot controlado por una computadora de realizar tareas comúnmente asociadas con seres inteligentes"* [5]

Si bien no existe una definición particular de la I.A., podemos observar que todas tienen un tema común y es la capacidad de que las computadoras puedan tener un comportamiento asociado a la definición de inteligencia.

Dentro de las distintas definiciones podemos encontrar cuatro grupos [6]. Unos miden el éxito en términos de fidelidad en la forma de actuar de los humanos, mientras que otros toman un concepto ideal de inteligencia denominado racionalidad. Un sistema es racional si hace "lo correcto", en función de su conocimiento. Los enfoques centrados en el comportamiento humano están basados en una ciencia empírica que incluyen hipótesis y confirmaciones mediante experimentos, mientras que el enfoque racional implica una

combinación de matemática e ingeniería. Dentro de los enfoques centrados en el comportamiento humano encontramos los sistemas que piensan como humanos, y aquellos que actúan como humanos. Mientras que dentro del enfoque de sistemas racionales encontramos los sistemas que piensan racionalmente, y los que actúan racionalmente.

2.2. Aprendizaje Automático

El aprendizaje automático puede ser definido de manera general como métodos computacionales que utilizan experiencia para mejorar el rendimiento o realizar predicciones acertadas. En este contexto la experiencia hace referencia a la información que poseía con anterioridad el sistema. En palabras de Mitchell (1997): *"Un programa de computación se dice que aprende de la experiencia E con respecto a algún tipo de tarea T y con medición del rendimiento P, si su rendimiento en la tarea T, medido por P, mejora con la experiencia E."*.

El tamaño y la calidad de esta información son cruciales para el acierto en las predicciones realizadas, entonces podemos decir que el aprendizaje automático consiste en el diseño de algoritmos predictivos que sean eficientes y precisos, dado que el éxito del algoritmo depende de los datos usados, el aprendizaje automático está inherentemente asociado al análisis de datos y la estadística.

Dentro del aprendizaje automático podemos encontrar distintos tipos de aprendizaje, como el aprendizaje supervisado y el no supervisado [7]. En el Aprendizaje supervisado, el sistema recibe información etiquetada como datos de entrenamiento y luego realiza predicciones sobre información no vista por el sistema. Mientras que en el Aprendizaje no supervisado, el sistema recibe información no etiquetada como datos de entrenamiento y luego realiza predicciones sobre información no vista por el sistema. Este tipo de aprendizaje está diseñado para generar conocimiento descubriendo regularidades en los datos, pero puede resultar difícil evaluar cuantitativamente el rendimiento del sistema. Como en este trabajo se utilizó el aprendizaje supervisado se explicará brevemente este tipo de aprendizaje.

Un algoritmo para el aprendizaje supervisado recibe como entrada el valor correcto (x) para determinados valores de una función ($f(x)$) desconocida y debe determinar cuál es la función (f) o aproximarla. Se podría decir que dada una colección de ejemplos de f, se debe devolver una función h que aproxime a f, $h(x) \simeq f(x)$. La función h se denomina hipótesis. Una buena hipótesis estará bien generalizada si puede predecir ejemplos que no conoce.

Dentro del aprendizaje supervisado podemos encontrar dos tipos de problemas donde se aplica: clasificación y regresión [8]. En los problemas de regresión se tienen un conjunto de datos de entrenamientos *D* de *N* puntos de entrenamiento (x_n, t_n), con $n = 1,\dots, N$, donde las variables x_n son las entradas y las variables t_n son las salidas deseadas (o target, como se las conoce en inglés), siendo estas variables continuas. El objetivo de la regresión es predecir la salida *t* para una entrada *x* no observada. Los problemas de clasificación son similares con la salvedad que la salida *t* representa valores discretos que toman un número finito de valores posibles, indicando la clase a la que pertenece la entrada *x*. Dentro de los problemas de clasificación encontramos dos tipos principales de clasificaciones: la clasificación binaria, donde sólo se pueden asignar dos clases diferentes (generalmente 0 o 1), o la clasificación multiclase, donde se pueden asignar múltiples categorías a la salida.

La calidad de la predicción $\hat{t}\,(x)$ para un par (x, t) se mide por una función de pérdida que podríamos llamar $l(t, \hat{t})$. Algunos ejemplos típicos de esta función incluyen la pérdida cuadrática $l(t, \hat{t}) = (t - \hat{t})^2$ para problemas de regresión y la tasa de error $l(t, \hat{t}) = 1(t \neq \hat{t})$, que equivale a 1 cuando la predicción es incorrecta y 0 de manera contraria, para problemas de clasificación. El objetivo del aprendizaje es minimizar la pérdida media en el par de prueba, la cual es referida como pérdida de generalización. Dicho de otra manera, se busca aumentar la capacidad de generalización del modelo. Es decir, que ante nuevos valores de *x*, no mostrados anteriormente al modelo, puedan encontrarse los valores $\hat{t}\,(x)$correctos.

En este trabajo como función de pérdida se optó por la función conocida como Categorical Cross-Entropy [9]. Para poder definir esta función primero debemos definir dos funciones, la función de activación Softmax y la función de pérdida Cross-Entropy. La función Softmax se define de la siguiente forma:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}}$$

Esta función se aplica a los valores obtenidos por la red, compacta dicho vector en el rango (0,1) y la suma de sus elementos da como resultado 1. Como cada elemento representa

una clase, la salida de esta función puede ser interpretada como una distribución categórica. La función de pérdida Cross-Entropy está dada por:

$$CE = -\sum_{i}^{C} t_i \log(s_i)$$

Donde t_i y s_i es el valor esperado y el obtenido por la red respectivamente. Con estos dos conceptos podemos realizar la definición de la función de pérdida Categorical Cross-Entropy. Dicha función es una concatenación entre las funciones descritas anteriormente, a los valores obtenidos por la red se le aplica la función Softmax y con dichos valores se calculan las pérdidas.

2.3. Aprendizaje Profundo

El aprendizaje profundo es una técnica del aprendizaje automático que utiliza redes neuronales complejas [10]. Éstas están inspiradas en la estructura del cerebro humano y se componen de un número de niveles jerárquicos. En el nivel inicial de la jerarquía la red aprende algo simple y luego envía esta información al siguiente nivel, que toma esta información, la combina, compone una información un poco más compleja y se lo pasa al tercer nivel, y así sucesivamente. Cada capa individual de la red puede ser pensada como un tipo de filtro que trabaja desde lo grueso a lo fino incrementando la probabilidad de obtener un resultado correcto. En general, las redes neuronales pueden realizar el mismo tipo de tareas que los algoritmos de aprendizaje automático clásicos. Pero no es posible afirmar lo mismo de manera inversa, las redes neuronales tienen capacidades que le permiten al aprendizaje profundo resolver tareas que los algoritmos clásicos de aprendizaje automático no podrán resolver.

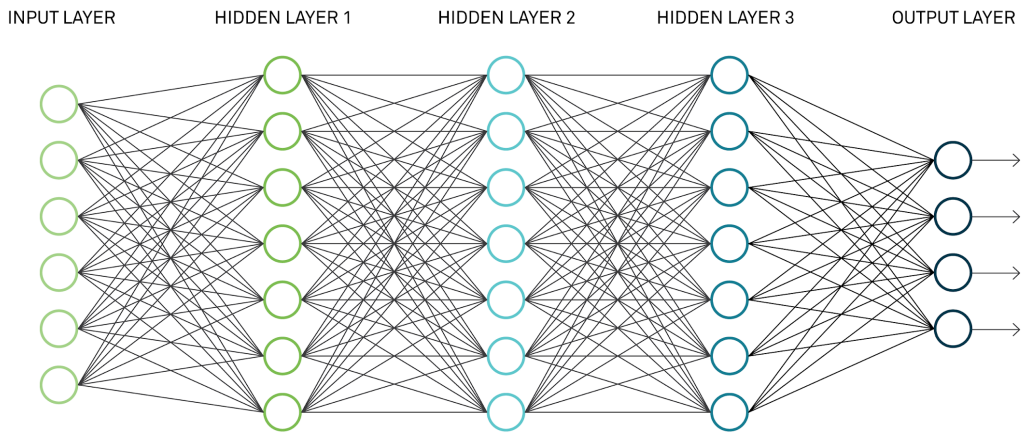

Figura 1: Red neuronal profunda[1]

[1] *https://www.directionsmag.com/article/8726*

2.4. Perceptrón multicapa

Una de las estructuras más simples de Redes Neuronales es el Perceptrón Multicapa o Multilayer Perceptron (MLP) , cuyo componente individual es el perceptrón [11].

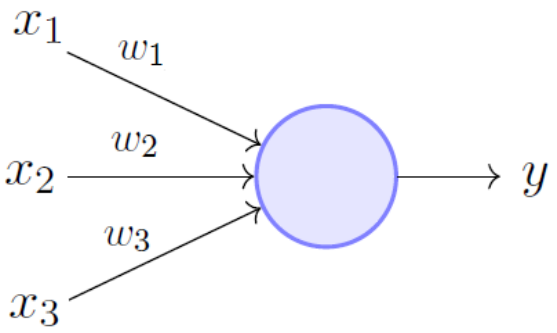

Figura 2: Modelo perceptrón - Minsky-Papert en 1996[2]

Como podemos ver en la figura 2, un perceptrón es un clasificador binario, compuesto por un vector de pesos $w = [w_1, \ldots, w_n]$, uno por cada entrada, más un peso *b* llamado Bias. Tanto *w* como *b* se los denomina parámetros. Con estos parámetros el perceptrón realiza el siguiente cálculo:

$$f(x) = \begin{cases} 1, \ si \ b + \sum_{i=1}^{l} x_i w_i > 0 \\ 0, \ de \ otra \ forma \end{cases}$$

Para simplificar la notación podemos utilizar la definición del producto punto entre dos vectores y es posible reescribir lo anterior como:

$$f(x) = \begin{cases} 1, \ si \ b + x.w > 0 \\ 0, \ de \ otra \ forma \end{cases}$$

f(x) es denominada como función de activación. Si bien en sus principios el perceptrón se pensó para resolver problemas lineales, en la actualidad es más frecuente encontrar problemas no lineales, los cuales no se pueden resolver con un perceptrón simple [11]

[2] *https://miro.medium.com/max/645/0*LJBO8UbtzK_SKMog*

El perceptrón multicapa es un modelo que se utiliza para resolver problemas no lineales, con aprendizaje supervisado. Su arquitectura se caracteriza por agrupar las neuronas (perceptrón simple) en capas, distinguiendo tres tipos de capas: la capa de entrada, la capa oculta y la capa de salida. La conexión entre capas puede ser total (Figura 3), es decir que cada neurona de la capa está conectada con todas las neuronas de la capa siguiente, o parcial, en la que cada neurona está conectada con algunas neuronas de la capa siguiente. Este tipo de redes son feedforward, esto quiere decir que el flujo de información comienza en las entradas y progresa a través de la red hasta la salida sin conexiones de feedback, es decir sin que el modelo se retroalimente a sí mismo.

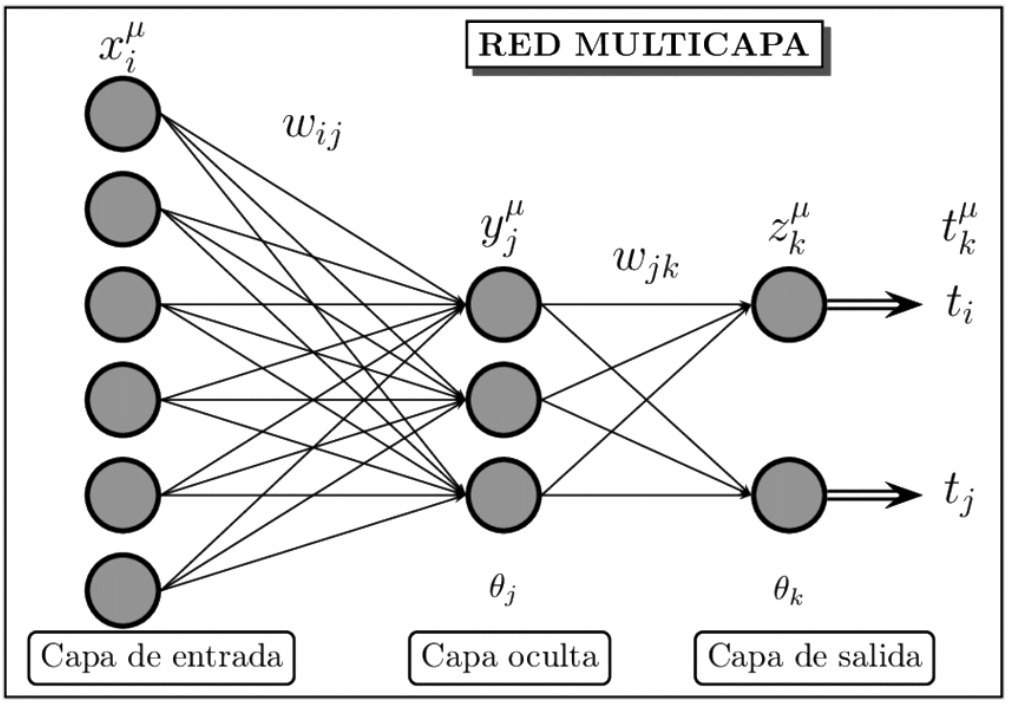

Figura 3: Conexión total[3]

La forma en la cual se ajusta el modelo es mediante algoritmos que modifican todos los parámetros de la red hasta que la salida sea similar a la deseada.

Como este modelo se utiliza en problemas de aprendizaje supervisado, para cada entrada se tiene la salida esperada, la cual es comparada con la salida obtenida mediante el entrenamiento del modelo, utilizando una función de error. En la mayoría de los casos la función de error está definida como

[3] *https://www.researchgate.net/profile/Javier-Aroztegui/publication/48932489/figure/fig16/AS:454465665933327@1485364466996/Figura-113-Perceptron-multicapa.png*

$$E = \frac{1}{N} \sum_{n=1}^{N} e(n)$$

Siendo *N* el número de muestras y *e* el error cometido por la red para la muestra *n*, este error está dado por

$$e(n) = \frac{1}{n_C} \sum_{i=1}^{n_C} \left(s_i(n) - y_i(n) \right)^2$$

donde *s(n)* e *y(n)* son las salidas, deseada y de la red respectivamente. Por lo tanto, se puede decir que el aprendizaje del modelo es equivalente a encontrar un mínimo de la función de error.

En modelos de aprendizaje automático el método más utilizado para encontrar este mínimo es el descenso de gradiente [12], mediante el cual calcula el gradiente como la derivada multivariable de la función de error. Para el cálculo de este gradiente se utiliza el algoritmo conocido como back-propagation [12], el cual consiste en empezar calculando las derivadas parciales de la función de error de la salida con respecto únicamente a los parámetros de la última capa. Luego, posicionados en la capa anterior, se calculan nuevamente las derivadas parciales, pero respecto a la capa actual y a lo obtenido en la modificación anterior. Este procedimiento continúa hasta llegar a la capa de entrada. Una vez obtenido el gradiente se lo multiplica por una tasa de aprendizaje, la cual nos permite ajustar la velocidad a la cual nos acercamos a la solución, y se actualizan los parámetros.

En este trabajo se empleó el algoritmo de optimización Adam [13], el cual es una extensión del algoritmo Descenso de Gradiente y que en la actualidad es más comúnmente utilizado. La principal diferencia entre ambos métodos es que mientras en el Descenso de Gradiente se tiene una única tasa de aprendizaje, con Adams se pueden tener tasas de aprendizajes individuales para diferentes parámetros estimando el primer y segundo momento del gradiente, siendo estos la media y la varianza no centrada respectivamente. La ventaja que encontramos en este algoritmo es su velocidad, ya que puede lograr resultados antes que otros algoritmos de optimización, como podemos ver en la figura 4.

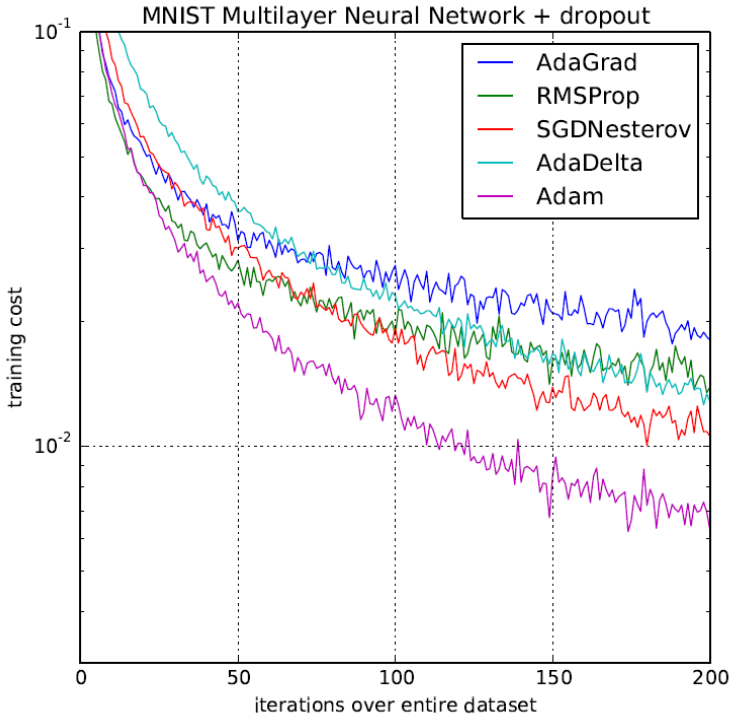

Figura 4: Comparación de Adams con otros algoritmos de optimización [13]

2.5. Redes Neuronales Convolucionales

También conocidas simplemente como redes convolucionales, las redes neuronales convolucionales (CNN – Convolutional Neural Networks) son un tipo especializado de redes neuronales para el procesamiento de datos con topología de red. Un ejemplo de este tipo de datos son las series de tiempo, las cuales pueden ser pensadas como una red de una dimensión (1D) tomando muestras a intervalos regulares de tiempo. Otro ejemplo de redes de dos dimensiones (2D) pueden ser imágenes donde la red está formada por los píxeles de la imagen. Como el nombre lo indica estas redes utilizan la operación matemática convolución, la cual se define como la integral del producto de dos funciones después de desplazar una de ellas una distancia *t*. *"CNN son simplemente redes neuronales que utilizan convolución en lugar de la multiplicación de matrices general en al menos una de sus capas"* [14]

La convolución se define de la siguiente manera:

$$s(t) = \int x(a)w(t - a)da$$

Esta operación se denomina regularmente con un asterisco $s(t) = (x * w)(t)$. En términos de las redes neuronales convolucionales el primer argumento (x) es referido frecuentemente como la entrada, y el segundo argumento (w), como el kernel. Al trabajar con datos digitales, el tiempo es discreto. Si asumimos que x y w están definidas únicamente en el dominio de los enteros podemos definir la convolución discreta como

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t - a)$$

En este tipo de aplicaciones la entrada, generalmente, es un array multidimensional de datos, y el kernel, usualmente un array multidimensional de parámetros que son modificados por el algoritmo de aprendizaje. Estos arrays multidimensionales se les suele dar el nombre de *tensor* debido a que cada elemento de la entrada y el kernel deben ser guardados explícitamente en lugares separados. Se suele asumir que estas funciones son cero en todo lugar salvo en el conjunto finito de puntos que tenemos como datos, por lo que es posible aplicar la sumatoria de un número infinito de puntos únicamente con los datos que se tienen almacenados. Si se trabaja con topologías de más de una dimensión se debe definir la convolución para tantas variables como tenga la topología.

La convolución aprovecha tres ideas que pueden ayudar a mejorar los sistemas de aprendizaje:

1. Interacciones dispersas: también conocida como conectividad dispersa o pesos dispersos, se logra haciendo al kernel más pequeño que la entrada haciendo que tengamos que almacenar menos parámetros, esto nos reduce los requerimientos de memoria del modelo y aumenta su eficiencia estadística. Esto nos permite que la red pueda describir eficientemente interacciones complicadas entre muchas variables construyendo dichas interacciones desde bloques sencillos.
2. Compartir parámetros: hace referencia al uso de los mismos parámetros para más de una función en un modelo. En una red tradicional cada elemento de la matriz de peso es usado una vez cuando se calcula la salida de la capa, en una red convolucional cada elemento del kernel es usado en cada posición de la entrada, pudiendo exceptuar los bordes dependiendo de las decisiones de diseño.
3. Representaciones equivariantes: debido a la forma en que se comparten parámetros, la capa obtiene una propiedad denominada equivariancia, lo que implica que si la

entrada cambia la salida cambia de la misma forma. Cuando se procesan series de tiempo la convolución genera una especie de línea de tiempo donde muestra cuando han aparecido distintas características en la entrada, si movemos un evento hacia delante en la entrada, la misma representación del evento aparecerá en la salida más tarde en el tiempo.

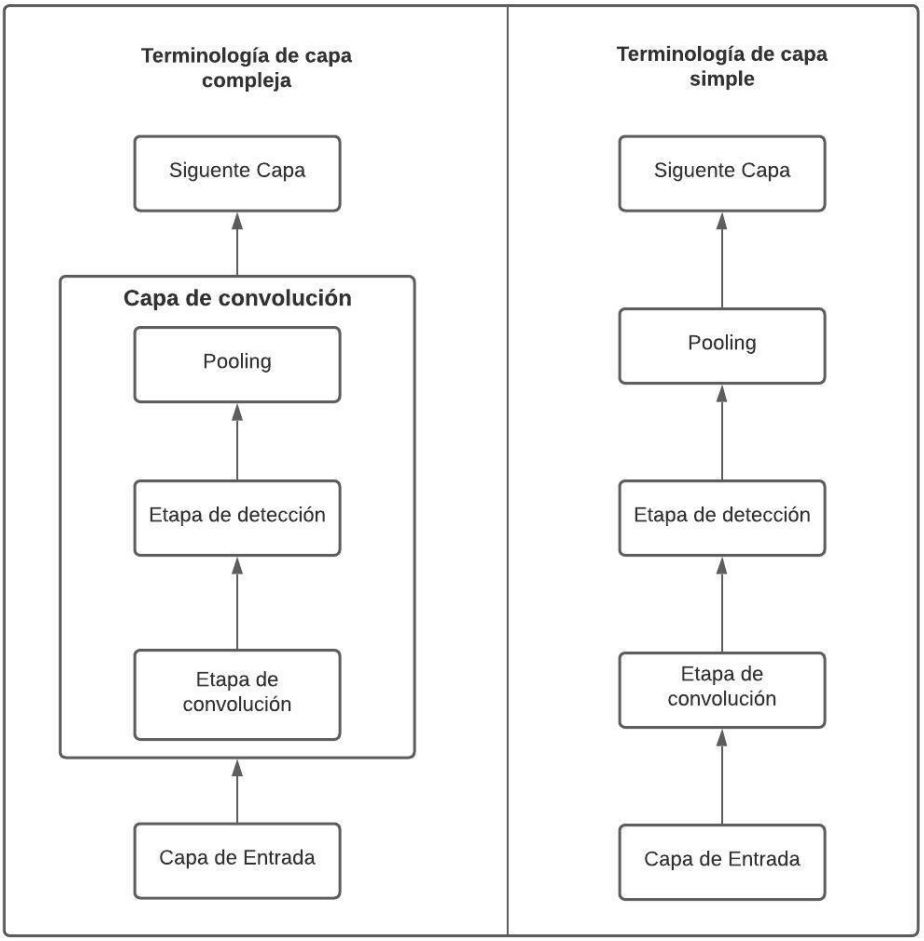

Figura 5: Terminología de capas convolucionales [14]

Una topología típica de una red convolucional consiste en tres capas. En la primera la capa realiza varias convoluciones en paralelo para producir un conjunto de activaciones lineales. En la segunda capa, cada activación lineal es pasada a una función de activación no lineal. En la tercera capa se usa una función de pooling para modificar la salida de la misma. Como podemos ver en la figura 5, existen dos tipos de terminologías en cuanto a las capas convolucionales, una terminología simple y otra compleja. La diferencia entre cada una radica en que para la terminología simple cada capa de la estructura convolucional es una capa en sí misma de menor complejidad.

La función de pooling aporta un mecanismo para reducir los mapas de características aplicando técnicas de resumen. Esto provoca que el mapa de características sea más robusto a los cambios de posición, es decir, que las representaciones se vuelvan invariantes a pequeños traslados en la entrada, haciendo que si trasladamos la entrada en un valor pequeño la salida no cambiará. Dos métodos comunes son el pooling promedio (Figura 7) y el pooling máximo (Figura 6), promediando las características y eligiendo la característica más representativa, respectivamente.

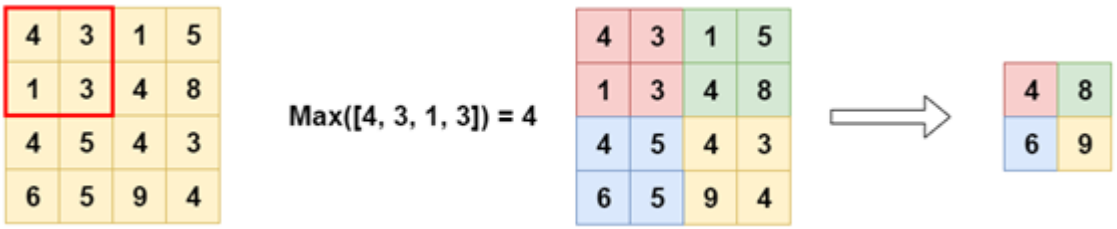

Figura 6: Pooling Máximo[4][5]

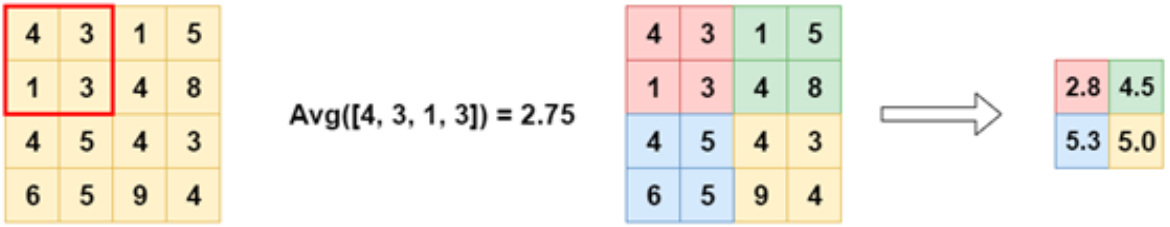

Figura 7: Pooling Promedio[6][7]

Para varios problemas a resolver, el pooling es esencial para manejar entradas de distintos tamaños. Por ejemplo, si queremos clasificar imágenes de distintos tamaños, la entrada de la capa de clasificación debe ser fija. Esto se realiza usualmente variando el tamaño de un offset entre las regiones de pooling así la capa de clasificación siempre recibe el mismo tamaño de entrada.

2.6. Optimalidad de Pareto

Los modelos de aprendizaje constan, como hemos visto en las subsecciones anteriores, de gran cantidad de hiperparámetros. Por ejemplo, en una red neuronal profunda, tenemos que definir la cantidad de capas, cantidad de neuronas en cada capa, funciones de activación,

[4] *https://www.machinecurve.com/wp-content/uploads/2020/01/Max-Pooling-1.png*
[5] *https://www.machinecurve.com/wp-content/uploads/2020/01/Max-Pooling-2.png*
[6] *https://www.machinecurve.com/wp-content/uploads/2020/01/Average-Pooling.png*
[7] *https://www.machinecurve.com/wp-content/uploads/2020/01/Average-Pooling-1.png*

entre tantos otros. Para poder obtener la mejor configuración de hiperparámetros de un modelo para un conjunto de datos determinado se utilizan algoritmos de optimización. Uno de éstos es el basado en el criterio de Optimalidad de Pareto.

Este criterio fue desarrollado por Vilfredo Pareto en su libro *"Manuale di economia politica"* publicado en 1906. Si bien es un concepto que viene de la economía tiene sus aplicaciones tanto en ingeniería como en distintas ciencias sociales. Una definición general de esta optimalidad es: Dada una asignación inicial de bienes entre un conjunto de individuos, se denomina mejora de Pareto una nueva asignación donde se mejore la situación de al menos un individuo sin perjudicar a otro y se refiere como Pareto óptima una asignación cuando no puede lograrse otra mejora de Pareto. Formalizando un poco más la definición podemos decir que sea P un problema de optimización multiobjetivo [15] entonces una solución de P es Pareto óptima cuando no exista otra solución tal que mejore un objetivo sin empeorar al menos uno de los otros objetivos. En general la solución en el sentido de Pareto al problema de optimización multiobjetivo no será única, todo este conjunto de soluciones se las conoce como Frontera de Pareto (Figura 8).

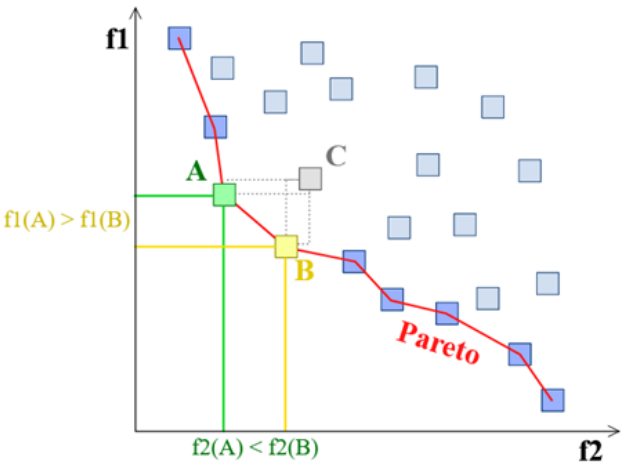

Figura 8: Frontera de Pareto[8]

Luego de la aplicación del algoritmo de optimización habremos obtenido un conjunto de soluciones a la optimización multiobjetivo, cada una de las cuales definirán los valores de los hiperparametros que optimicen la red neuronal. Como se puede ver en la figura 8, cada conjunto solución minimiza un objetivo distinto como podría ser la precisión de la predicción

[8] *https://es.wikipedia.org/wiki/Eficiencia_de_Pareto#/media/Archivo:Front_pareto.svg*

contra la velocidad en la que se realiza la misma, esto nos permite seleccionar el conjunto solución que mejor se adapte al objetivo planteado.

Los conceptos teóricos presentados en esta sección son la base de los modelos a desarrollar en las próximas secciones. La Inteligencia Artificial desde sus comienzos, nos permite comprender el núcleo de las redes neuronales que sirven de punto de partida del desarrollo del modelo de Aprendizaje Profundo basado en Redes Neuronales Convolucionales y en los Modelos Perceptrón Multicapa. También se hará uso de la función de pérdida Categorical Cross-Entropy y de Adam como algoritmo de optimización. Luego, el criterio planteado por Pareto nos permite obtener la mejor configuración posible del modelo propuesto optimizando su funcionamiento sin pérdida de precisión.

3. Metodologías, herramientas y tecnologías

Por tratarse de un proyecto de investigación, la metodología a aplicar debió tener la flexibilidad necesaria para adaptar cambios y ajustes en las metas parciales y los tiempos de planificación. A su vez, fue necesario un marco de trabajo que permita realizar las revisiones y evaluaciones de resultados por parte de los directores de proyecto, a fin de verificar los avances que permitieran la continuación a nuevas etapas.

Metodología Ad-hoc

Durante el proyecto se utilizó una metodología Ad-Hoc cíclica (Figura 9), donde cada ciclo se puede dividir en dos etapas principales. La primera es la etapa de planificación, la cual se subdivide en dos actividades: análisis y presentación. La segunda es la etapa de realización, que involucra tres actividades: investigación, desarrollo y validación. Cada ciclo duró 15 días aproximadamente, siendo el inicio de éste el día donde se realizaba la reunión con los directores del proyecto. A lo largo del proyecto fueron necesarios 7 ciclos.

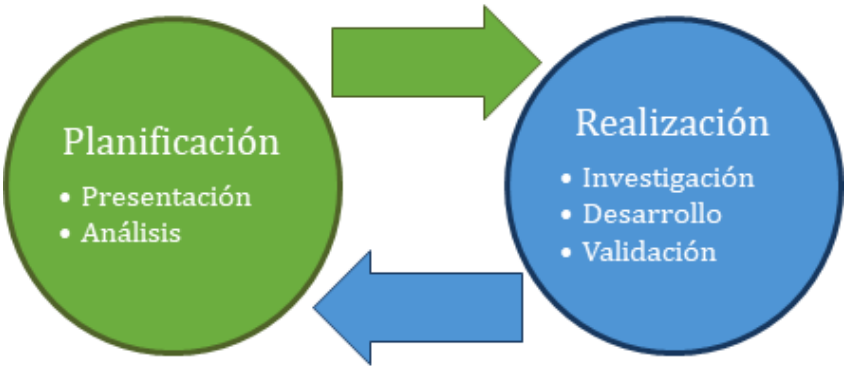

Figura 9: Metodología Ad-hoc cíclica.

La etapa de planificación comenzaba con una reunión del equipo de trabajo con los directores del proyecto. En esta reunión se presentaban los trabajos y resultados logrados en las etapas previas para que sean analizados por los directores, quienes verificaban los avances alcanzados. A partir de este análisis se definía si era necesario continuar en el ciclo actual porque aún restaban trabajos por concluir, o si podía pasarse al siguiente ciclo del proyecto asignando a cada integrante las tareas a realizar.

Durante la etapa de realización se llevaban a cabo los trabajos asignados en la planificación, existiendo comunicación con los directores del proyecto en caso de dudas o problemas. Esta etapa estaba definida por diferentes actividades que podían ser de investigación, desarrollo o validación según el estado del proyecto.

Durante las primeras iteraciones, la realización se basaba principalmente en la investigación y análisis de material sobre el dominio, búsqueda de conjuntos de datos y modelos de aprendizaje profundo para el estudio y determinación de su utilidad en el proyecto. En forma se cumplían los objetivos de investigación y análisis, y se reunían los resultados necesarios para la continuación del proyecto, se daba comienzo al diseño y desarrollo del prototipo. Una vez que estuvo avanzado el desarrollo comenzaron a realizarse las pruebas y recolección de los resultados.

Herramientas y tecnologías utilizadas.

Para el desarrollo de los diferentes Scripts de Pyhton a lo largo del proyecto se utilizó Google colaboratory (Colab), que es un entorno que permite escribir código y ejecutarlo en la nube. Para poder hacer esto se basa en el concepto de Notebook implementado por Jupyter[9]

[9] *https://jupyter.org/*

pero almacenado en Google Drive. Estas Notebooks se componen principalmente de dos tipos de celdas. Una de estas puede contener código mientras que la restante puede contener texto, imágenes, etc. Al ejecutarse en los servidores de Google nos da acceso a un entorno virtual con el cual solo es necesario una conexión a internet y un navegador web para poder comenzar a programar. En este entorno se pueden cargar datos, no solo desde una fuente local sino también desde la nube. Esto nos permite garantizar la consistencia de los recursos durante el desarrollo ya que cualquier actualización de los mismos se verá reflejada en la plataforma. También permite a Colab una forma fácil de poder compartir las Notebooks creadas, mediante Google Drive, copiando el código a una cuenta de GitHub o descargando el archivo, el cual se guarda en el formato establecido por las Notebook Jupyter.

Como base para el desarrollo de las notebooks mediante Colab y para la posterior realización del informe hemos utilizado Google Drive, que es una plataforma de almacenamiento de datos en la nube. Cada usuario cuenta con 15Gb de espacio gratuito pudiendo ampliar este límite, se puede acceder a través de su sitio web y dispone de aplicaciones para el acceso a través de dispositivos móviles. Posee integración y da soporte al resto de los servicios de la empresa como Google Colaboratory y su paquete de ofimática entre otros.

Para acceder a los datasets utilizados en el proyecto mediante Colab de una forma sencilla sin el requerimiento de otorgar permisos a usuarios que quieran ejecutar el código, hemos utilizado Dropbox. Al igual que Google Drive, es una plataforma que permite almacenar datos en la nube y posee distintos planes según la necesidad.

Para el desarrollo de todo el proyecto se ha elegido Python como lenguaje de programación. Éste es un lenguaje interpretado, dinámico y multiplataforma. También es multiparadigma ya que soporta el paradigma orientado a objetos, la programación imperativa y la programación funcional. La filosofía en la que se desarrolló este lenguaje gira entorno a la legibilidad de su código mediante una sintaxis sencilla que se asemeja más al lenguaje humano. Para la administración de memoria Python utiliza tipado dinámico y conteo de referencias. Esta última es una técnica para saber cuántas veces un recurso está siendo referido. Otra facilidad que ofrece Python es el desarrollo de nuevos módulos mediante la programación en C o C++. Además de estas ventajas, el lenguaje fue elegido por la amplia variedad y disponibilidad de librerías que ofrecen implementaciones de los modelos neuronales de interés para el desarrollo del proyecto.

El análisis de datos y la generación de modelos se ha realizado mediante Keras, la que es una API escrita en Python que corre sobre la plataforma de aprendizaje automático TensorFlow. También provee abstracciones y bloques de código que permiten el desarrollo de soluciones de aprendizaje automático de manera ágil. [16]. Las estructuras de datos centrales en Keras son capas y modelos, siendo el modelo más sencillo el secuencial que está compuesto por una pila lineal de capas. Keras también permite el desarrollo de modelos enteros desde cero. Además sigue el principio del descubrimiento progresivo, lo que hace fácil comenzar el desarrollo y también permite el manejo arbitrario de casos de usos avanzados con el único requisito de un aprendizaje incremental a cada paso. A continuación, mencionamos algunas de las capas que se han utilizado en el proyecto con una descripción acerca de que realiza cada una de ellas:

Conv1D: Es una capa de convolución en una dimensión, esta capa crea un kernel de convolución y se realiza la convolución con la capa de entrada produciendo un tensor como salida.

MaxPooling1D: Esta capa trabaja en una dimensión y realiza una reducción de la entrada, tomando el valor máximo de una ventana que es definida mediante un parámetro al momento de creación de la capa.

GlobalAveragePooling1D: Es una capa que realiza una reducción de la entrada mediante una operación de pooling promedio.

Dropout: Aplica la operación de dropout a la entrada, ignorando ciertas neuronas durante la fase de entrenamiento de un conjunto de neuronas que son elegidas al azar. El ser ignoradas significa que estas unidades no son tenidas en cuenta durante la fase de Forward o Backward en el proceso de entrenamiento.

Dense: Este tipo de capas es una capa tradicional densamente conectada de una red neuronal. Una capa está densamente conectada si cada neurona de la capa está conectada con todas las neuronas de la capa anterior.

4 . Desarrollo

En esta sección se detalla cómo se cumplieron los distintos objetivos específicos planteados a lo largo del desarrollo de este informe de proyecto. En primer lugar, se detalla el análisis de la utilización de modelos de aprendizaje maquinal, especialmente los llamados profundos, en el mantenimiento predictivo de la industria. Además, se detalla la obtención y análisis de conjuntos de datos utilizados en el proyecto. Posteriormente, se da cuenta del proceso de la selección y evaluación de modelos de aprendizaje profundo para el tratamiento de los datos seleccionados y que servirán de core para el prototipo de herramienta propuesto. También se detalla el diseño y desarrollo de este prototipo de herramienta de software que permite reunir todas estas técnicas, artefactos y modelos propuestos para tareas de mantenimiento predictivo. Por último, se da cuenta de las pruebas en tiempo real que permiten evaluar el funcionamiento del prototipo y analizar los resultados.

4.1. Revisión de la literatura

Como inicio de nuestro proyecto de investigación se comenzó con la búsqueda y análisis de diferentes trabajos, proyectos y papers que permitan conocer el estado actual de la utilización de modelos de aprendizaje maquinal para aplicación en mantenimiento predictivo de la industria. En especial, se intentó descubrir la utilización de modelos de aprendizaje profundo para esta tarea.

Existen en la actualidad gran cantidad de trabajos, algunos en desarrollo, en una amplia gama de campos dentro de la industria, tales como: Marítima, Ferroviaria, Aeroespacial, de Producción eléctrica, Petroleras y Manufacturera [2]. En estos trabajos hemos observado la aplicación de una serie de modelos, algunos de los cuales son: Inferencia difusa [17], K-medias [17], Modelos estadísticos [18], Modelos probabilísticos [19], Máquinas de vector soporte (en inglés: Support Vector Machines) [20] y Redes Neuronales Artificiales (en inglés: Artificial Neural Network) [21].

Debido a la naturaleza multidisciplinaria de la actividad de mantenimiento industrial donde deben trabajar profesionales de diferentes áreas, encontramos en algunos proyectos la creación de entornos de trabajo. En estos entornos se busca abstraer la complejidad de los posibles modelos, aplicándolos en diferentes artefactos. A estos artefactos se accede por medio de distintas interfaces, por un lado los ingenieros encargados del diseño, mantenimiento y actualización de los modelos, y por otro los encargados del mantenimiento

de los equipos. Rescatamos la importancia de que cada usuario tenga interfaces con lenguaje y datos que pueda interpretar, desde el ingeniero hasta el operador del equipo industrial. [22]

Por otro lado, si bien desde el comienzo de este trabajo estamos hablando de analizar la gran cantidad de datos que se generan mediante el uso de modelos de aprendizaje profundo, lo cierto es que nos encontramos con la dificultad de obtener en la industria grandes conjuntos de datos etiquetados. Los diferentes equipos industriales poseen sensores que acumulan grandes repositorios con series de datos, pero en la mayoría de los casos estas series de datos carecen de las etiquetas de estado del equipo, necesarias para el proceso de entrenamiento de los modelos.

Una forma de atacar este problema es realizando un trabajo de etiquetado recurriendo al conocimiento experto del dominio del equipo que ha sido monitoreado. Es por esto que se decidió realizar una búsqueda de conjuntos de datos que hayan sido utilizados y validados en investigaciones anteriores y cuenten con el etiquetado necesario.

4.2. Identificación de conjuntos de datos existentes

Para la búsqueda de conjuntos de datos aplicables al trabajo se utilizaron distintas plataformas que hubieran sido utilizadas o mencionadas en la literatura analizada. A continuación se detallan los conjuntos de datos seleccionados durante esta etapa del proyecto.

Los conjuntos de datos

La Universidad de California (Riverside) posee un repositorio público de conjuntos de datos (datasets) de series de tiempo ya etiquetadas para su utilización en trabajos de clasificación[10]. De este repositorio seleccionamos dos conjuntos de datos que por su naturaleza, eran aplicables al proyecto. Estos datasets se utilizaron originalmente en una competencia en el Congreso Mundial de la IEEE sobre Inteligencia Computacional en 2008. El problema de clasificación consistía en diagnosticar la existencia o no de problemas en un subsistema automotriz. Cada muestra comprende 500 mediciones de ruido del motor y una clasificación binaria. Hay dos datasets distintos, el primero denominado FordA en el que el conjunto de datos de entrada posee una contaminación acústica mínima; y el segundo denominado FordB en el que las series de tiempo de entrada poseen ruido.

[10] *http://www.timeseriesclassification.com*

Características de los conjuntos de datos	
Cantidad de Muestras (pruebas)	500
Cantidad de Atributos (series de tiempo)	1
Cantidad de Componentes con condición etiquetada.	1

Tabla 1: Características de los conjuntos de datos FordA y FordB.

Por otro lado, la Universidad de California (Irvine) también dispone de un repositorio[11] donde se publican gran cantidad de conjuntos de datos con el fin de fomentar la investigación de las distintas técnicas de aprendizaje automático.

De este repositorio obtuvimos un conjunto de datos con series de tiempo que provienen de distintos sensores ubicados en una plataforma de pruebas hidráulicas y las etiquetas con los estados de cuatro diferentes componentes luego de transcurrido el tiempo de las mediciones reportadas por los sensores.[23]

El mismo conjunto de datos fue encontrado en el repositorio Kaggle, una subsidiaria de Google LLC, esta es una comunidad en línea de científicos de datos y profesionales del aprendizaje automático. Kaggle comenzó en 2010 ofreciendo concursos de aprendizaje automático y ahora también pone a disposición una plataforma de datos pública, un banco de trabajo basado en la nube para ciencia de datos y educación en inteligencia artificial. En sus repositorios encontramos y seleccionamos el mismo conjunto de datos que descargamos del repositorio de la Universidad de California (Irvine) y que provenía de una plataforma de pruebas hidráulicas.

Características del conjunto de datos	
Cantidad de Muestras (pruebas)	2205
Cantidad de Atributos (series de tiempo)	17
Cantidad de Componentes con condición etiquetada.	5

Tabla 2: Características del conjunto de datos de monitoreo de sistema hidráulico.

[11] *https://archive.ics.uci.edu/ml/index.php*

Selección del conjunto de datos para el proyecto

Luego de la lectura de la documentación y fuentes de los conjuntos de datos encontrados en los diferentes repositorios decidimos comenzar a realizar pruebas con el dataset de Monitoreo de Plataforma Hidráulica de Pruebas. El motivo para esta selección fue que se trata de un conjunto de datos más amplio que los demás, debido a que tiene mayor cantidad de muestras, de atributos y componentes con condición etiquetada.

Estos datos, atributos y objetivos monitoreados nos permiten realizar una mayor cantidad de pruebas exploratorias sobre el conjunto de datos con el fin de encontrar subconjuntos de atributos y etiquetas para analizarlos y probarlos en un modelo de aprendizaje profundo.

Descripción del conjunto de datos seleccionado

A continuación se describe con más detalle el conjunto de datos seleccionado para el proyecto. Estos datos abordan la evaluación del estado de un banco de pruebas hidráulico basado en datos de distintos tipos de sensores. Se superponen cuatro tipos de fallas con varios niveles de gravedad.

El sistema repite ciclos de carga constante (durante 60 segundos), en cada ciclo realiza mediciones con distintas frecuencias y al finalizar este registra la condición de cuatro componentes.

El conjunto de datos tiene en total 2205 muestras, por cada muestra los 17 sensores generan series de datos a distinta frecuencia en la entrega de lecturas. Entonces la cantidad de atributos totales se da sumando la cantidad de lecturas que aportan todos los sensores durante el minuto que dura la prueba:

	8 sensores x 60 lecturas (1 Hz)
+	2 sensores x 600 lecturas (10 Hz)
+	7 sensores x 6000 lecturas (100 Hz)
Total	**43680 atributos**

En la tabla 3 se pueden apreciar en detalle las características de los distintos atributos y en la tabla 4 las distintas etiquetas de salida con los posibles valores.

Información de los atributos

Sensor	Tipo	Unidad	Frecuencia de muestreo	Archivo
PS1	Presión	bar	100 Hz	PS1.txt
PS2	Presión	bar	100 Hz	PS2.txt
PS3	Presión	bar	100 Hz	PS3.txt
PS4	Presión	bar	100 Hz	PS4.txt
PS5	Presión	bar	100 Hz	PS5.txt
PS6	Presión	bar	100 Hz	PS6.txt
EPS1	Potencia	W	100 Hz	EPS1.txt
FS1	Flujo	l/min	10 Hz	FS1.txt
FS2	Flujo	l/min	10 Hz	FS2.txt
TS1	Temperatura	ºC	1 Hz	TS1.txt
TS2	Temperatura	ºC	1 Hz	TS2.txt
TS3	Temperatura	ºC	1 Hz	TS3.txt
TS4	Temperatura	ºC	1 Hz	TS4.txt
VS1	Vibración	mm/s	1 Hz	VS1.txt
CE	Eficiencia de enfriado	%	1 Hz	CE.txt
CP	Potencia de enfriado	kW	1 Hz	CP.txt
SE	Factor de eficiencia	%	1 Hz	SE.txt

Tabla 3:Características de los atributos del conjunto de datos.

Valores de las condiciones de los distintos componentes

Objetivo	Unidad	Valor	Significado
Condición del enfriador	%	3	Próximo a falla total.
		20	Eficiencia reducida.
		100	Eficiencia total.

Condición de la válvula	%	73	Próximo a falla total.
		80	Retraso severo.
		90	Retraso leve.
		100	Comportamiento óptimo.
Pérdida de la bomba		2	Pérdida severa.
		1	Pérdida leve.
		0	Sin pérdida.
Acumulador	bar	90	Próximo a falla total.
		100	Presión severamente reducida.
		115	Presión levemente reducida.
		130	Presión óptima.
Estabilidad		1	Condiciones variables.
		0	Condiciones estables.

Tabla 4:Características de las etiquetas del conjunto de datos.

Análisis exploratorio del conjunto de datos

Una vez seleccionado el conjunto de datos con el que trabajar en el proyecto, realizamos una análisis exploratorio de datos donde leemos, seleccionamos, graficamos y calculamos ciertas medidas estadísticas del conjunto para lograr conocer las principales características de las series de tiempo.

Para este análisis, realizado en Python, tomamos porciones del conjunto de datos, agrupando los sensores de acuerdo a su tipo.

En la Figura 10 podemos observar las curvas correspondientes a los cuatro sensores de temperatura en cuatro diferentes muestras, mientras en la Figura 11 se observa la totalidad de las muestras de los sensores de temperatura concatenadas en el orden que las contiene el conjunto de datos.

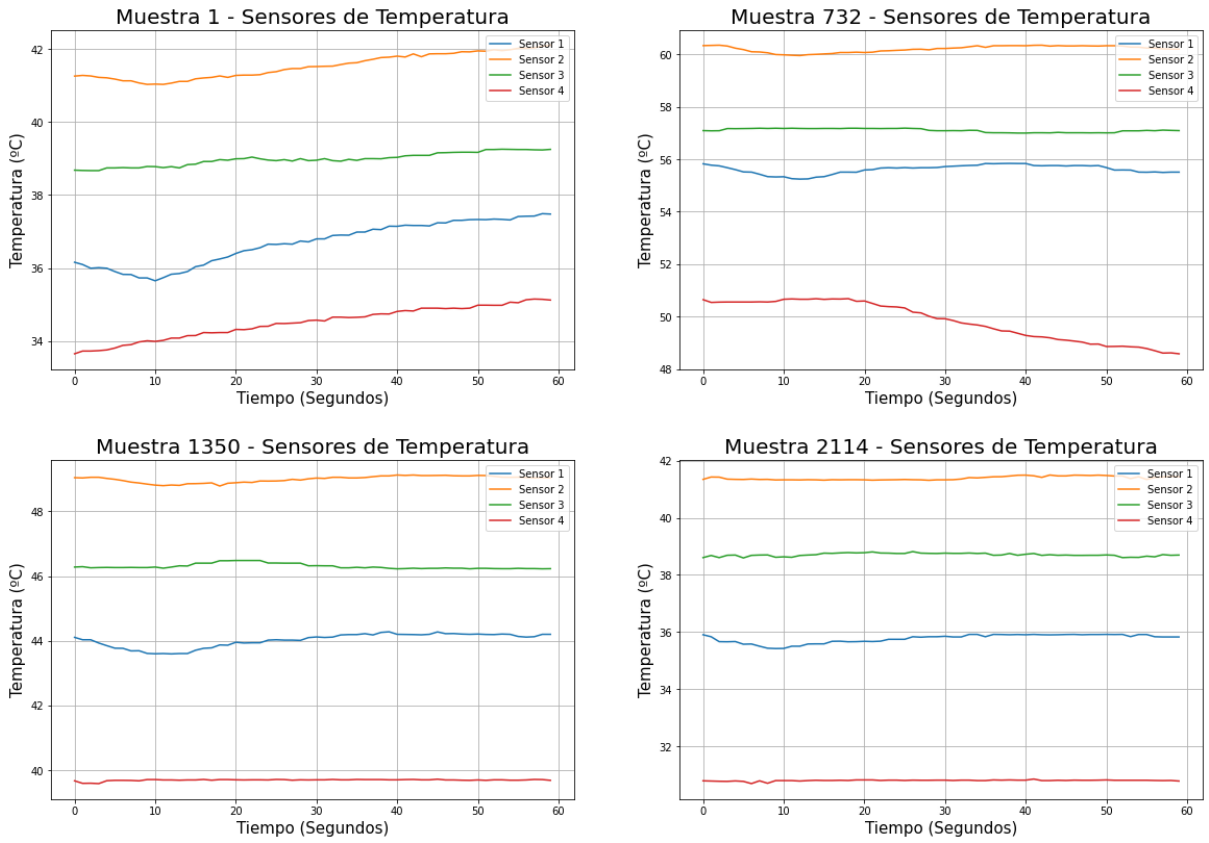

Figura 10: Gráficas de las muestras de sensores de temperatura seleccionadas aleatoriamente.

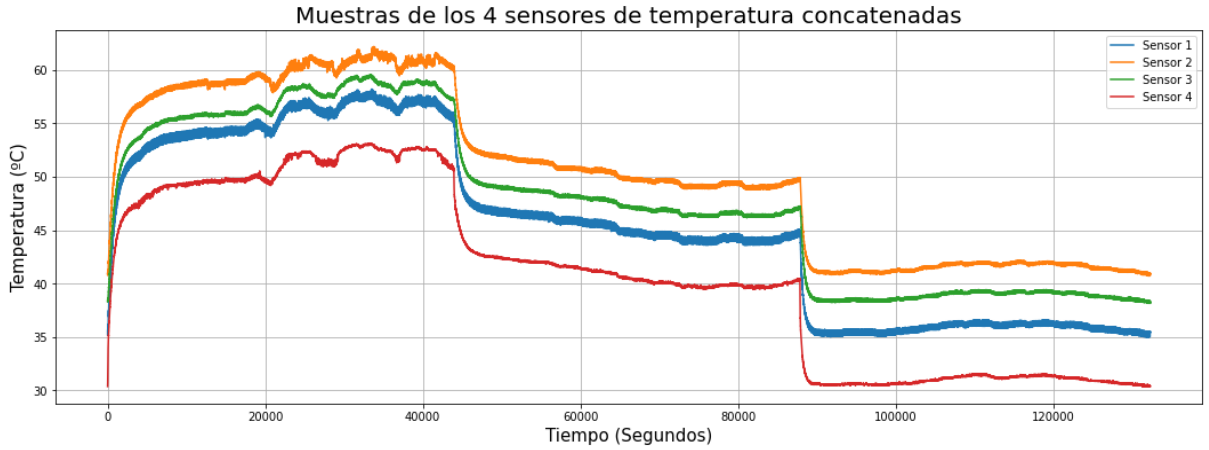

Figura 11: Las 2205 muestras de los sensores de temperatura concatenadas.

En la figura 12 se grafica el comportamiento de los seis sensores de presión en cuatro muestras diferentes y la figura 13 muestra las curvas de los sensores de presión en las diez primeras muestras concatenadas.

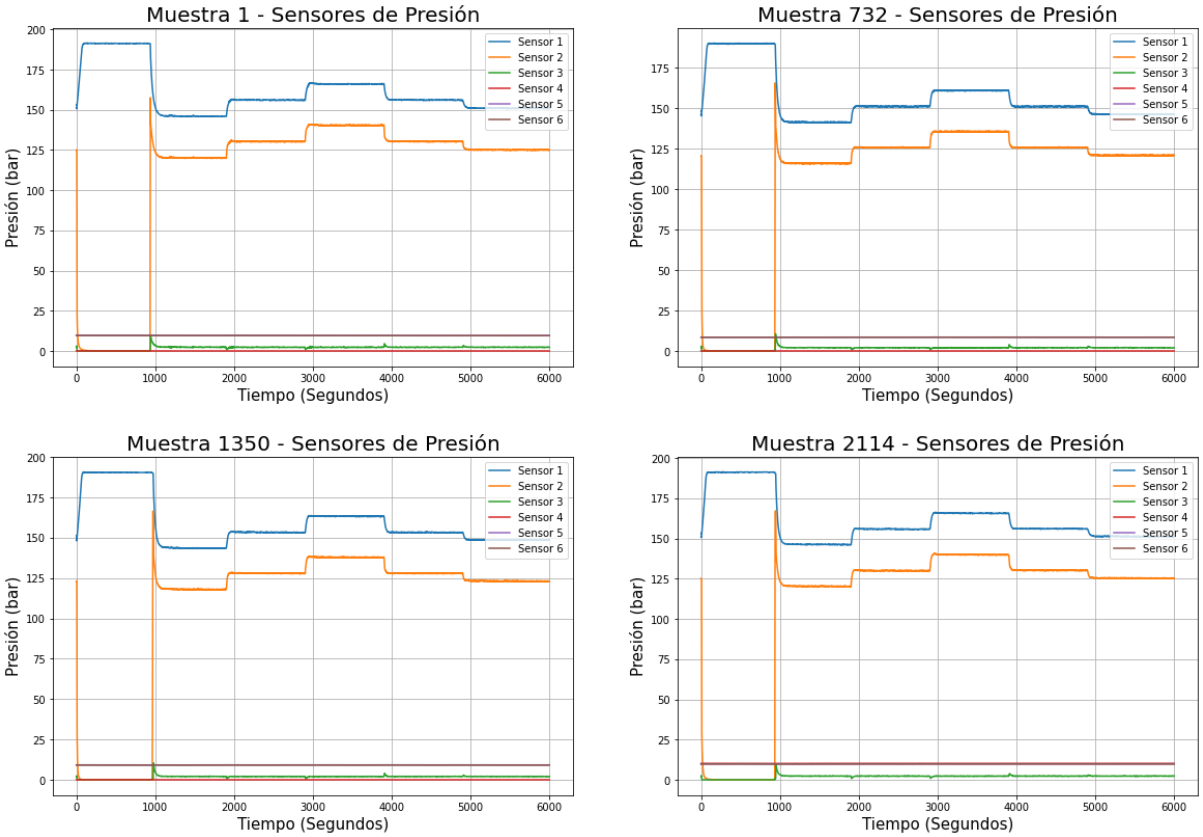

Figura 12: Gráficas de las muestras de sensores de presión seleccionadas aleatoriamente.

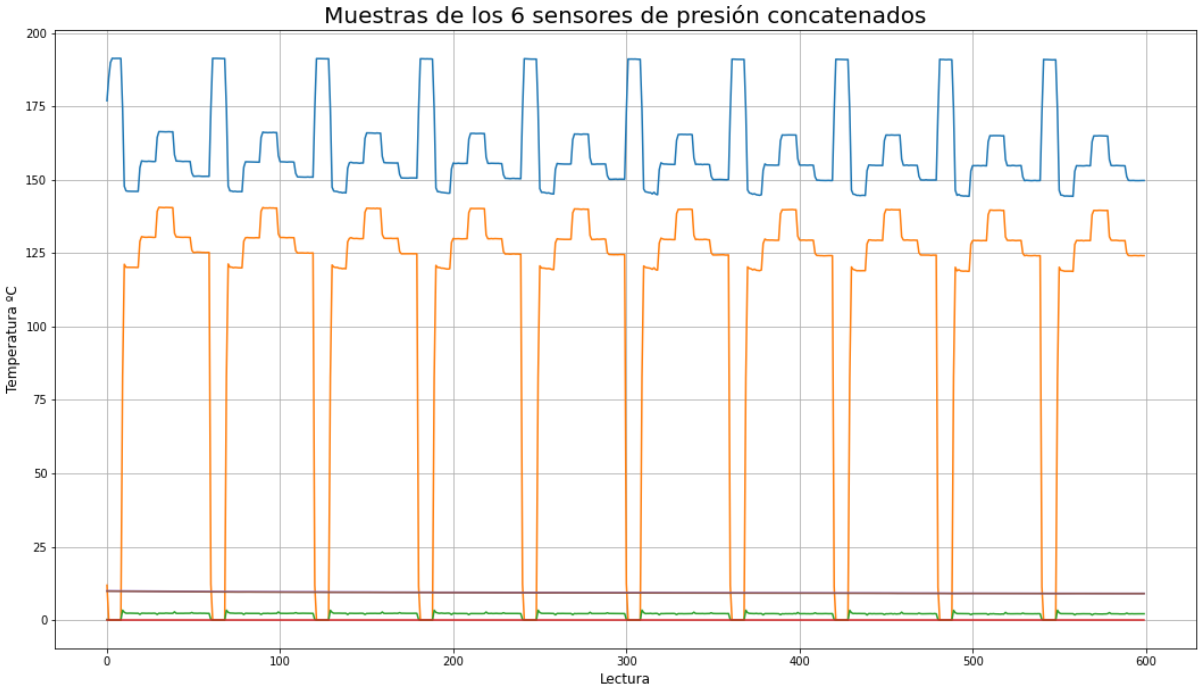

Figura 13: Las primeras 10 muestras de los sensores de presión concatenadas.

Se observan en la figura 14 las curvas correspondientes a los dos sensores de flujo en cuatro muestras distintas y en la figura 15 se muestran las diez primeras muestras de los sensores de flujo concatenadas.

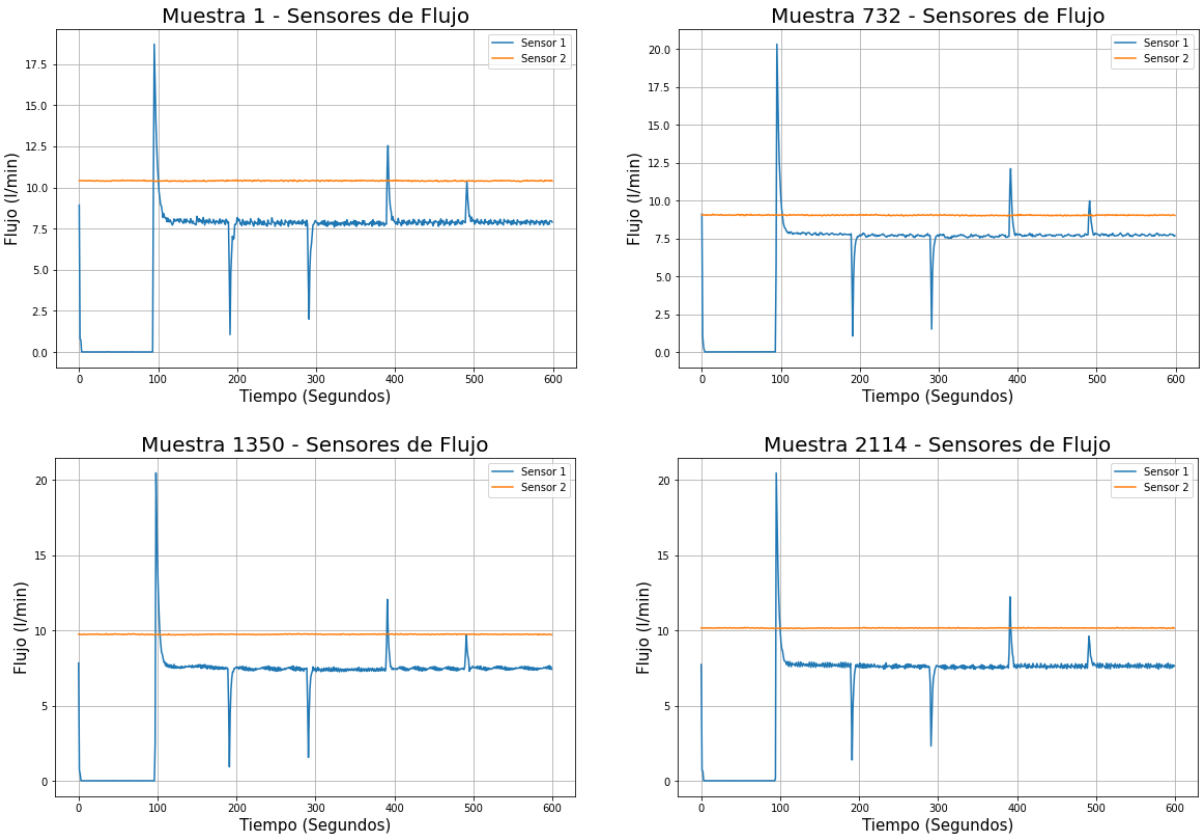

Figura 14: Gráficas de las muestras de sensores de flujo seleccionadas aleatoriamente.

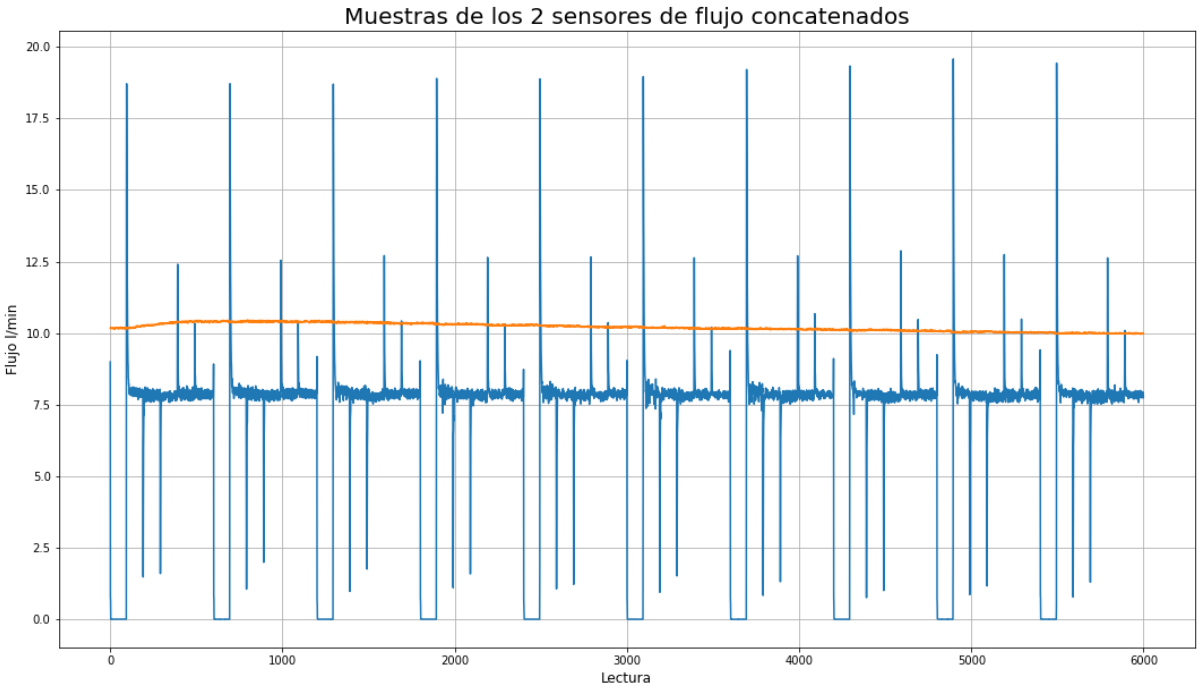

Figura 15: Las primeras 10 muestras de los sensores de flujo concatenadas.

Muestra 1	
Objetivo	**Estado**
Enfriador	Próximo a falla total
Valvula	Comportamiento óptimo
Bomba	Sin pérdida
Acumulador	Presión óptima
Estabilidad	Condiciones variables

Muestra 732	
Objetivo	**Estado**
Enfriador	Eficiencia reducida
Valvula	Comportamiento óptimo
Bomba	Sin pérdida
Acumulador	Próximo a falla total
Estabilidad	Condiciones variables

Muestra 1350	
Objetivo	**Estado**
Enfriador	Eficiencia reducida
Valvula	Próximo a falla total
Bomba	Pérdida severa
Acumulador	Próximo a falla total
Estabilidad	Condiciones estables

Muestra 2114	
Objetivo	**Estado**
Enfriador	Eficiencia total
Valvula	Comportamiento óptimo
Bomba	Pérdida leve
Acumulador	Próximo a falla total
Estabilidad	Condiciones variables

Tabla 5: Etiquetas de estado de los componentes de las muestras seleccionadas aleatoriamente.

En la tabla 5 se muestran las cinco etiquetas de salida del conjunto de datos para cinco muestras distintas elegidas al azar.

Balance del conjunto de datos

En cuanto a las etiquetas de salida, en la Tabla 6 visualizamos el balance en la distribución del conjunto de datos con respecto a cada componente o salida.

Componente / Salida	**Estado**	**Cantidad de muestras**
Enfriador	Próximo a falla total.	732
	Eficiencia reducida.	732
	Eficiencia total.	741
Válvula	Próximo a falla total.	360
	Retraso severo.	360
	Retraso leve.	360
	Comportamiento óptimo.	1125
Pérdida en la bomba	Pérdida severa.	492
	Pérdida leve.	492
	Sin pérdida.	1221

Acumulador	Próximo a falla total.	808
	Presión severamente reducida.	399
	Presión levemente reducida.	399
	Presión óptima.	599
Estabilidad	Condiciones variables.	756
	Condiciones estables.	1449

Tabla 6: Distribución de muestras para las distintas etiquetas de salida.

4.3. Selección de modelos de aprendizaje profundo

Dentro del gran dominio de la Industria 4.0 encontramos que se utilizan distintos modelos de aprendizaje profundo, en diferentes áreas. La selección del modelo a utilizar depende de muchos factores: la naturaleza de los datos, la existencia de conjuntos de datos etiquetados, la cantidad de datos disponibles, entre otros.

En este trabajo las fuentes de datos son sensores dispuestos en diferentes partes de un equipo que con cierta periodicidad estable y conocida reportan una medición que describe alguna variable (temperatura, presión, flujo, nivel, etc.), por lo que estas fuentes tienen la forma de Series de Tiempo. Esta característica de los datos con los que se trabaja hace necesario, para poder aplicar algún tipo de modelo de aprendizaje supervisado que permita clasificar el estado del equipamiento o componente, la existencia de conjuntos de datos con Series de Tiempo etiquetadas con el estado del componente o equipo que se está censando para poder entrenar el modelo de aprendizaje profundo.

Dentro de los modelos que permiten trabajar con Series de Tiempo existen dos grandes grupos:

- Redes Neuronales Recurrentes que utilizan para predecir tanto el estado actual de las entradas como el valor de las predicciones pasadas.
- Redes Neuronales Convolucionales que mediante filtros permiten identificar características para luego en conjunto lograr la predicción.

Para la realización de este trabajo se optó por modelos convolucionales teniendo en cuenta que el objetivo del modelo de aprendizaje profundo sería reconocer diferentes

características en las señales, entregadas como series de tiempo, que permitan clasificar un estado del equipamiento o componente en el futuro.

Selección de subconjuntos de datos

Debido a la gran cantidad y variedad de datos del dataset seleccionado para realizar pruebas, definimos subconjuntos que trabajen con un mismo tipo de sensor de entrada y un objetivo al cual clasificar su estado. Esta selección se realizó deduciendo que características medidas afectarían el estado de un determinado componente.

Así se seleccionaron cuatro nuevos conjuntos de datos para las pruebas:

Nombre	Sensores	Frec. de lectura	Pasos de tiempo por muestra	Componente
FS1-2_LEAKING	Flujo (FS1 y FS2)	10Hz	600	Bomba.
PS1-6_LEAKING	Presión (PS1, PS2, PS3, PS4, PS5 y PS6)	100Hz	6000	Bomba.
PS1-6_VALVE	Presión (PS1, PS2, PS3, PS4, PS5 y PS6)	100Hz	6000	Válvula.
TS1-4_COOLER	Temperatura (TS1, TS2, TS3 y TS4)	1Hz	60	Cooler.

Tabla 7: Características originales de los cuatros subconjuntos seleccionados.

Con el fin de trabajar con el mismo formato en los conjuntos de datos para poder simplificar las pruebas y el modelo, consumir menos tiempo de cómputo y así poder realizar más experimentos, se modificaron los conjuntos de entrada de los sensores de flujo y presión que tienen 600 y 6000 pasos de tiempo por muestra respectivamente creando para estos series de 60 pasos de tiempo tomando la media cada 10 lecturas en el caso de los sensores de flujo y cada 100 lecturas en el caso de los sensores de presión.

Luego de esta modificación los conjuntos de datos seleccionados para las pruebas quedaron como muestra la siguiente tabla:

Nombre	Sensores	Pasos por muestra	Componente
FS1-2_LEAKING	Flujo (FS1 y FS2)	60	Bomba.

PS1-6_LEAKING	Presión (PS1, PS2, PS3, PS4, PS5 y PS6)	60	Bomba.
PS1-6_VALVE	Presión (PS1, PS2, PS3, PS4, PS5 y PS6)	60	Válvula.
TS1-4_COOLER	Temperatura (TS1, TS2, TS3 y TS4)	60	Cooler.

Tabla 8: Características finales de los cuatros subconjuntos seleccionados.

El modelo seleccionado

Para las pruebas con el conjunto de datos seleccionado se utilizó inicialmente un modelo convolucional 1D simple. La estructura del mismo se extrajo y ajustó directamente de la documentación de Keras, el framework de Python seleccionado para implementar el modelo. La red neuronal con la que se realizaron las pruebas iniciales sobre el conjunto de datos seleccionado tiene la siguiente estructura (se indica la línea en python que define las capas):

1) Capa convolucional 1D con 100 filtros de tamaño 6 y función de activación Relu.

```
model.add(layers.Conv1D(100, 6, activation='relu', input_shape=(60, X*)))
```

2) Capa convolucional 1D con 100 filtros de tamaño 6 y función de activación Relu.

```
model.add(layers.Conv1D(100, 6, activation='relu'))
```

3) Capa MaxPooling 1D con tamaño de ventana 3.

```
model.add(layers.MaxPooling1D(3))
```

4) Capa convolucional 1D con 100 filtros de tamaño 6 y función de activación Relu.

```
model.add(layers.Conv1D(100, 6, activation='relu'))
```

5) Capa convolucional 1D con 100 filtros de tamaño 6 y función de activación Relu.

```
model.add(layers.Conv1D(100, 6, activation='relu'))
```

6) Capa GlobalAveragePooling1D.

```
model.add(layers.GlobalAveragePooling1D())
```

7) Capa Dropout con una tasa de 0,5.

```
model.add(layers.Dropout(0.5))
```

8) Capa de salida, completamente conectada, con tres unidades de salida, una por cada clase y función de activación Softmax.

```
model.add(layers.Dense(y, activation='softmax'))
```

Como se puede observar inicialmente se utilizaron cien filtros de características en cada capa convolucional y el tamaño del kernel (ventana convolucional) es 6, el 10% del largo de la

muestra en pasos de tiempo. Con esta configuración las cantidades de parámetros entrenables por capa y totales quedan expresadas en las siguientes tablas:

Modelo para el conjunto FS1-2_LEAKING (input: 60 pasos, 2 series)(out: 3 clases)	
Capa	**Cantidad de Parámetros Entrenables**
1er. convolucional 1D	1300
2er. convolucional 1D	60100
3er. convolucional 1D	60100
4er. convolucional 1D	60100
Salida completamente conectada	303
TOTAL PARÁMETROS ENTRENABLES	181903

Tabla 9: Parámetros entrenables por capa y totales para el modelo del conjunto FS1-2_LEAKING.

Modelo para el conjunto PS1-6_LEAKING (input: 60 pasos, 6 series)(out: 3 clases)	
Capa	**Cantidad de Parámetros Entrenables**
1er. convolucional 1D	3700
2er. convolucional 1D	60100
3er. convolucional 1D	60100
4er. convolucional 1D	60100
Salida completamente conectada	303
TOTAL PARÁMETROS ENTRENABLES	184303

Tabla 10: Parámetros entrenables por capa y totales para el modelo del conjunto PS1-6_LEAKING.

Modelo para el conjunto PS1-6_VALVE (input: 60 pasos, 6 series)(out: 4 clases)	
Capa	**Cantidad de Parámetros Entrenables**
1er. convolucional 1D	3700
2er. convolucional 1D	60100
3er. convolucional 1D	60100
4er. convolucional 1D	60100

Salida completamente conectada	404
TOTAL PARÁMETROS ENTRENABLES	184404

Tabla 11: Parámetros entrenables por capa y totales para el modelo del conjunto PS1-6_VALVE.

Modelo para el conjunto TS1-4_COOLER (input: 60 pasos, 4 series)(out: 3 clases)	
Capa	**Cantidad de Parámetros Entrenables**
1er. convolucional 1D	2500
2er. convolucional 1D	60100
3er. convolucional 1D	60100
4er. convolucional 1D	60100
Salida completamente conectada	303
TOTAL PARÁMETROS ENTRENABLES	183103

Tabla 12: Parámetros entrenables por capa y totales para el modelo del conjunto TS1-4_COOLER.

Para el entrenamiento del modelo se utilizó entropía cruzada como función de pérdida y Adam como algoritmo de optimización.

Estructura de los scripts para entrenamiento del modelo seleccionado

Para la utilización del modelo seleccionado y las variantes que se fueron aplicando a lo largo del proyecto se utilizó siempre una estructura similar en el script . Esto permitió reutilizar el código y minimizar las pérdidas de tiempo en corrección de errores y depuración, la Figura 16 da cuenta de los pasos que contiene dicha estructura.

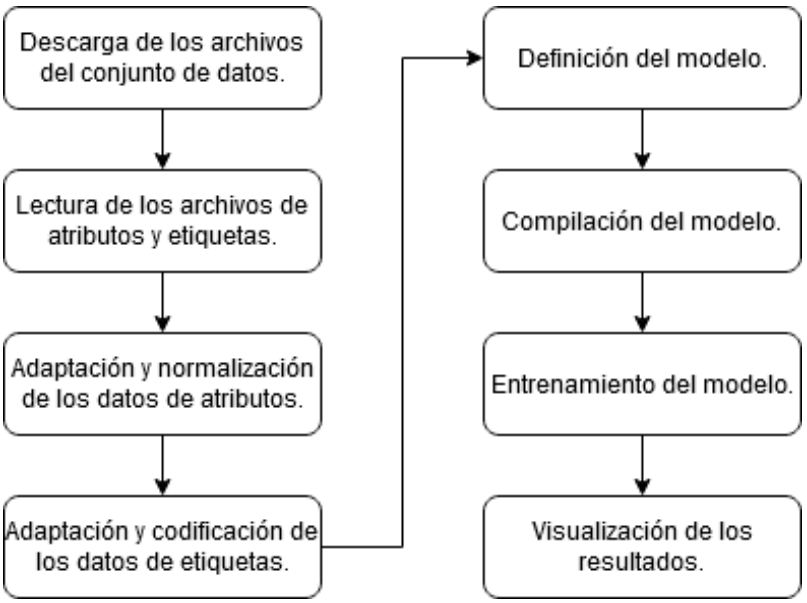

Figura 16: Etapas dentro de los scripts que prueban los modelos.

Como primer paso descargamos los archivos que comprenden el conjunto de datos, que se encuentran alojados en un repositorio en la nube (en este caso, Dropbox). De los archivos se leen las distintas estructuras de datos utilizando la librería Pandas, siendo necesario normalizar los datos de los atributos a valores entre 0 y 1. Esto se realiza , por un lado para mejorar la convergencia del entrenamiento y, por otro lado, para adaptar la forma de los distintos vectores y matrices, a fin de que puedan ser interpretados por la capa de entrada de la red neuronal. También se divide el conjunto de entrada en dos partes, para entrenamiento y test. Los datos de las etiquetas deben ser codificados con una codificación One-hot para que pueda ser interpretada la salida del modelo y comparar esta con el resultado esperado, expresado en la etiqueta.

Con el conjunto de datos ya preprocesado, se define el modelo de la red neuronal con todas sus capas, y luego se compila especificando la función de error, método de ajuste del entrenamiento y la métrica para seguir la evolución del proceso. Por último se realiza el entrenamiento del modelo especificando el tamaño de lote, la cantidad de épocas y la porción de datos utilizada como conjunto de validación. Se pueden visualizar los resultados de precisión alcanzada, su evolución a lo largo del proceso de entrenamiento y los valores de la función de error. También se evalúa la precisión alcanzada con el conjunto de test.

Pruebas con el conjunto de datos FS1-2_LEAKING

Durante el entrenamiento del modelo de red neuronal implementado en un script de Python[12], Keras tiene la opción de salida que nos informa luego de cada época los valores alcanzados de pérdida y precisión con el conjunto de entrenamiento y con el conjunto de validación (Figura 17).

```
fit_history = model.fit(X_train, y_train, batch_size=256, epochs=30, validation_split=0.2, verbose=2)

Epoch 1/30
5/5 - 2s - loss: 1.0414 - accuracy: 0.5057 - val_loss: 0.9993 - val_accuracy: 0.5275
Epoch 2/30
5/5 - 1s - loss: 0.9791 - accuracy: 0.5583 - val_loss: 0.9925 - val_accuracy: 0.5275
Epoch 3/30
5/5 - 1s - loss: 0.9651 - accuracy: 0.5640 - val_loss: 0.9722 - val_accuracy: 0.5275
Epoch 4/30
5/5 - 1s - loss: 0.9558 - accuracy: 0.5794 - val_loss: 0.9643 - val_accuracy: 0.5955
Epoch 5/30
5/5 - 1s - loss: 0.9370 - accuracy: 0.5972 - val_loss: 0.9452 - val_accuracy: 0.5955
Epoch 6/30
5/5 - 1s - loss: 0.9257 - accuracy: 0.5989 - val_loss: 0.9411 - val_accuracy: 0.5955
Epoch 7/30
5/5 - 1s - loss: 0.9114 - accuracy: 0.5964 - val_loss: 0.9030 - val_accuracy: 0.5955
Epoch 8/30
5/5 - 1s - loss: 0.8864 - accuracy: 0.6078 - val_loss: 0.9164 - val_accuracy: 0.6117
Epoch 9/30
5/5 - 1s - loss: 0.8971 - accuracy: 0.5981 - val_loss: 0.8677 - val_accuracy: 0.5955
Epoch 10/30
5/5 - 1s - loss: 0.8709 - accuracy: 0.5981 - val_loss: 0.8829 - val_accuracy: 0.5955
Epoch 11/30
5/5 - 1s - loss: 0.8528 - accuracy: 0.6183 - val_loss: 0.8366 - val_accuracy: 0.5955
Epoch 12/30
5/5 - 1s - loss: 0.8378 - accuracy: 0.6062 - val_loss: 0.8231 - val_accuracy: 0.6570
Epoch 13/30
5/5 - 1s - loss: 0.8172 - accuracy: 0.6240 - val_loss: 0.8050 - val_accuracy: 0.6570
Epoch 14/30
5/5 - 1s - loss: 0.7983 - accuracy: 0.6353 - val_loss: 0.7894 - val_accuracy: 0.6699
Epoch 15/30
5/5 - 1s - loss: 0.7991 - accuracy: 0.6386 - val loss: 0.7751 - val accuracy: 0.6570
Epoch 16/30
5/5 - 1s - loss: 0.7845 - accuracy: 0.6532 - val_loss: 0.8151 - val_accuracy: 0.5955
Epoch 17/30
5/5 - 1s - loss: 0.7892 - accuracy: 0.6402 - val_loss: 0.7482 - val_accuracy: 0.6570
Epoch 18/30
5/5 - 1s - loss: 0.7619 - accuracy: 0.6661 - val_loss: 0.7845 - val_accuracy: 0.5922
Epoch 19/30
5/5 - 1s - loss: 0.7958 - accuracy: 0.6288 - val_loss: 0.7417 - val_accuracy: 0.6861
Epoch 20/30
5/5 - 1s - loss: 0.7587 - accuracy: 0.6791 - val_loss: 0.7345 - val_accuracy: 0.6570
Epoch 21/30
5/5 - 1s - loss: 0.7328 - accuracy: 0.6994 - val_loss: 0.7292 - val_accuracy: 0.6861
Epoch 22/30
5/5 - 1s - loss: 0.7468 - accuracy: 0.7018 - val_loss: 0.7235 - val_accuracy: 0.7055
Epoch 23/30
5/5 - 1s - loss: 0.7383 - accuracy: 0.7172 - val_loss: 0.7502 - val_accuracy: 0.5566
Epoch 24/30
5/5 - 1s - loss: 0.7696 - accuracy: 0.6507 - val_loss: 0.7257 - val_accuracy: 0.6537
Epoch 25/30
5/5 - 1s - loss: 0.7314 - accuracy: 0.6799 - val_loss: 0.7163 - val_accuracy: 0.6602
Epoch 26/30
5/5 - 1s - loss: 0.7238 - accuracy: 0.7018 - val_loss: 0.7140 - val_accuracy: 0.6570
Epoch 27/30
5/5 - 1s - loss: 0.7310 - accuracy: 0.6969 - val_loss: 0.7275 - val_accuracy: 0.6990
Epoch 28/30
5/5 - 1s - loss: 0.7376 - accuracy: 0.6710 - val_loss: 0.7487 - val_accuracy: 0.6893
Epoch 29/30
5/5 - 1s - loss: 0.7382 - accuracy: 0.6807 - val_loss: 0.7657 - val_accuracy: 0.6634
Epoch 30/30
5/5 - 1s - loss: 0.7655 - accuracy: 0.6677 - val_loss: 0.7146 - val_accuracy: 0.7023
```

Figura 17: Captura de pantalla de los resultados del entrenamiento con 30 épocas y 20% de datos para validar.

[12] *https://colab.research.google.com/drive/1FPqOm76TjCTJso-TfHGyuxW6ud4sYoO_?usp=sharing*

Para este conjunto de datos se extendió el número de épocas hasta 30, buscando que se estabilicen los valores de pérdida y precisión indicando que se alcanzó el mejor resultado posible con esa configuración.

Luego de finalizado el proceso de entrenamiento, procedemos a graficar la evolución de los valores de pérdida y precisión durante las épocas del proceso para apreciar el comportamiento de estos valores y verificar si las curvas convergen a un valor estable (Figura 18).

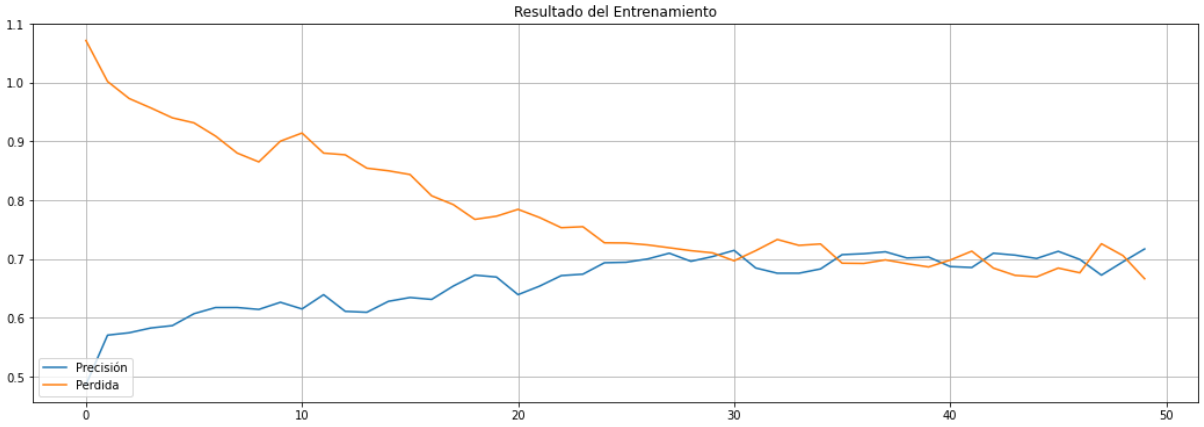

Figura 18: Gráfica de la evolución de precisión y pérdida durante el entrenamiento.

Al evaluar el modelo entrenado utilizando el 30% del conjunto de muestras reservado para test, arrojó una precisión de 72,51%.

Pruebas con el conjunto de datos PS1-6_LEAKING

Para el conjunto de datos PS1-6_LEAKING se implementó el modelo[13] y se utilizaron diez épocas en el entrenamiento no lográndose una precisión mejor al 53.4% con el conjunto de validación (Figura 19), luego se graficó la evolución durante las épocas del valor de la pérdida y precisión (Figura 20).

[13] *https://colab.research.google.com/drive/1Gvfb43BHSeK8SXz-kj8fsG-Mnk2JqEYF?usp=sharing*

```
fit_history = model.fit(X_train, y_train, batch_size=256, epochs=10, validation_split=0.2, verbose=2)

Epoch 1/10
5/5 - 2s - loss: 1.0388 - accuracy: 0.5130 - val_loss: 1.0240 - val_accuracy: 0.5340
Epoch 2/10
5/5 - 1s - loss: 1.0090 - accuracy: 0.5689 - val_loss: 1.0205 - val_accuracy: 0.5340
Epoch 3/10
5/5 - 1s - loss: 0.9976 - accuracy: 0.5697 - val_loss: 1.0238 - val_accuracy: 0.5340
Epoch 4/10
5/5 - 1s - loss: 1.0016 - accuracy: 0.5697 - val_loss: 1.0195 - val_accuracy: 0.5340
Epoch 5/10
5/5 - 1s - loss: 0.9970 - accuracy: 0.5697 - val_loss: 1.0239 - val_accuracy: 0.5340
Epoch 6/10
5/5 - 1s - loss: 1.0021 - accuracy: 0.5697 - val_loss: 1.0180 - val_accuracy: 0.5340
Epoch 7/10
5/5 - 1s - loss: 0.9978 - accuracy: 0.5697 - val_loss: 1.0209 - val_accuracy: 0.5340
Epoch 8/10
5/5 - 1s - loss: 0.9905 - accuracy: 0.5697 - val_loss: 1.0167 - val_accuracy: 0.5340
Epoch 9/10
5/5 - 1s - loss: 0.9934 - accuracy: 0.5697 - val_loss: 1.0144 - val_accuracy: 0.5340
Epoch 10/10
5/5 - 1s - loss: 0.9891 - accuracy: 0.5697 - val_loss: 1.0136 - val_accuracy: 0.5340
```

Figura 19: Captura de pantalla de los resultados del entrenamiento con 10 épocas y 20% de datos para validar.

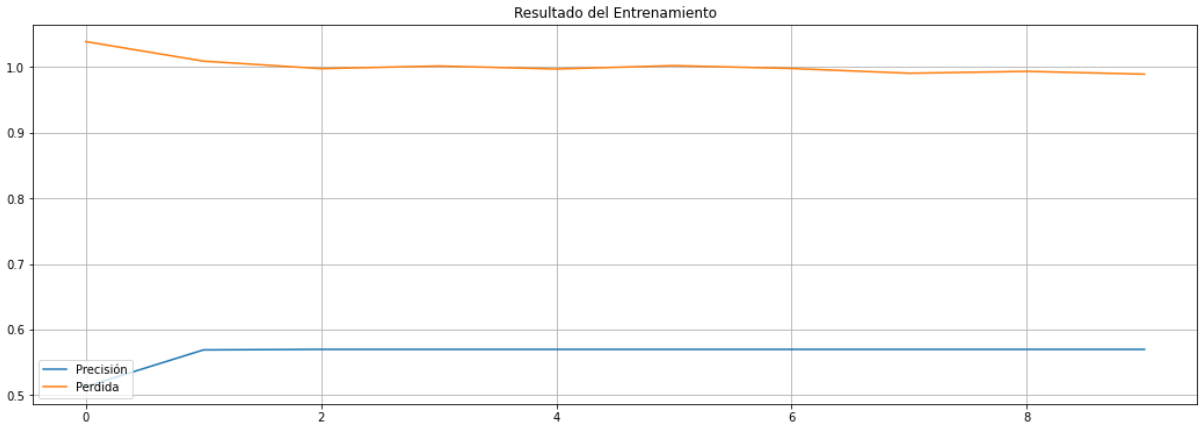

Figura 20: Gráfica de la evolución de precisión y pérdida durante el entrenamiento.

Al evaluar el modelo, luego del entrenamiento, con el conjunto de test arrojó una precisión de 53,32%.

Pruebas con el conjunto de datos PS1-6_VALVE

El entrenamiento del conjunto PS1-6_VALVE se realizó[14] también con diez épocas alcanzando una precisión del 100% con el conjunto de validación (Figura 21), luego se graficó el comportamiento de los valores de pérdida y precisión (Figura 22).

[14] *https://colab.research.google.com/drive/1RY1ePd_yrCTPHwOX8v3VAIKlFvh-1DIB?usp=sharing*

```
fit_history = model.fit(X_train, y_train, batch_size=16, epochs=10, validation_split=0.2, verbose=2)

Epoch 1/10
78/78 - 3s - loss: 1.2629 - accuracy: 0.5081 - val_loss: 1.2487 - val_accuracy: 0.5081
Epoch 2/10
78/78 - 1s - loss: 1.2543 - accuracy: 0.5138 - val_loss: 1.2961 - val_accuracy: 0.5081
Epoch 3/10
78/78 - 1s - loss: 1.2500 - accuracy: 0.5138 - val_loss: 1.2373 - val_accuracy: 0.5081
Epoch 4/10
78/78 - 1s - loss: 1.2342 - accuracy: 0.5138 - val_loss: 1.2337 - val_accuracy: 0.5081
Epoch 5/10
78/78 - 1s - loss: 1.2320 - accuracy: 0.5138 - val_loss: 1.2547 - val_accuracy: 0.5081
Epoch 6/10
78/78 - 1s - loss: 1.0819 - accuracy: 0.5551 - val_loss: 0.7448 - val_accuracy: 0.7055
Epoch 7/10
78/78 - 1s - loss: 0.5673 - accuracy: 0.7974 - val_loss: 0.3083 - val_accuracy: 0.9191
Epoch 8/10
78/78 - 1s - loss: 0.3182 - accuracy: 0.8987 - val_loss: 0.1901 - val_accuracy: 0.8964
Epoch 9/10
78/78 - 1s - loss: 0.2121 - accuracy: 0.9392 - val_loss: 0.0656 - val_accuracy: 1.0000
Epoch 10/10
78/78 - 1s - loss: 0.1086 - accuracy: 0.9814 - val_loss: 0.0525 - val_accuracy: 1.0000
```

Figura 21: Captura de pantalla de los resultados del entrenamiento con 10 épocas y 20% de datos para validar.

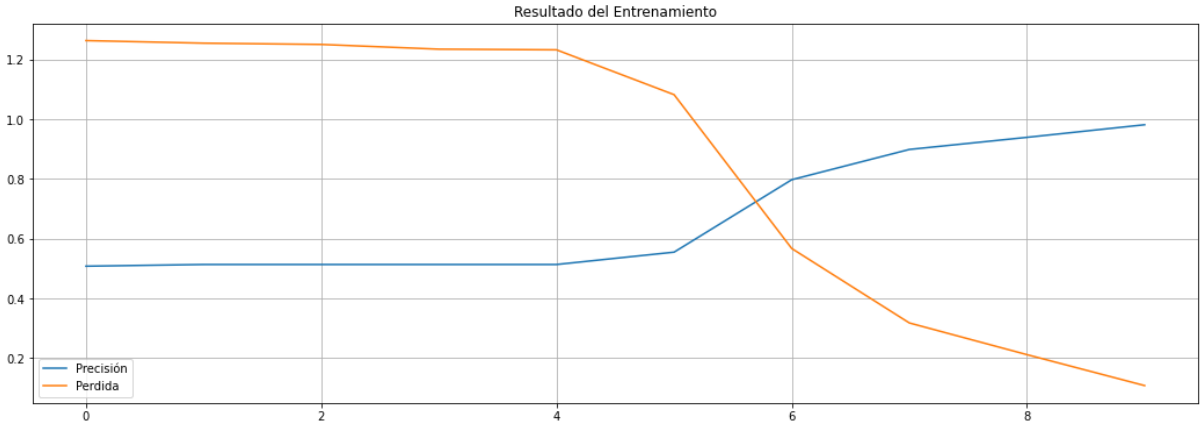

Figura 22: Gráfica de la evolución de precisión y pérdida durante el entrenamiento.

Al evaluar el modelo, luego del entrenamiento, con el conjunto de test arrojó una precisión de 100%.

Pruebas con el conjunto de datos TS1-4_COOLER

Para el conjunto TS1-4_COOLER también fueron suficientes diez épocas[15] para alcanzar valores estables de pérdida y precisión durante el entrenamiento alcanzando una precisión del 99.35% con el conjunto de validación (Figura 23). Luego graficamos la evolución de los valores de pérdida y precisión durante el proceso de entrenamiento (Figura 24).

[15] *https://colab.research.google.com/drive/1JgvutsQfvZl1tkVo0sGxbJVtLuLtMHrI?usp=sharing*

```
fit_history = model.fit(X_train, y_train, batch_size=16, epochs=10, validation_split=0.2, verbose=2)

Epoch 1/10
78/78 - 3s - loss: 0.4637 - accuracy: 0.7771 - val_loss: 0.0437 - val_accuracy: 0.9935
Epoch 2/10
78/78 - 2s - loss: 0.1309 - accuracy: 0.9619 - val_loss: 0.2333 - val_accuracy: 0.8835
Epoch 3/10
78/78 - 1s - loss: 0.1097 - accuracy: 0.9692 - val_loss: 0.0272 - val_accuracy: 0.9935
Epoch 4/10
78/78 - 1s - loss: 0.0777 - accuracy: 0.9789 - val_loss: 0.0915 - val_accuracy: 0.9806
Epoch 5/10
78/78 - 1s - loss: 0.1134 - accuracy: 0.9652 - val_loss: 0.0215 - val_accuracy: 0.9935
Epoch 6/10
78/78 - 1s - loss: 0.0869 - accuracy: 0.9773 - val_loss: 0.0177 - val_accuracy: 0.9935
Epoch 7/10
78/78 - 1s - loss: 0.1136 - accuracy: 0.9660 - val_loss: 0.0408 - val_accuracy: 0.9935
Epoch 8/10
78/78 - 1s - loss: 0.0830 - accuracy: 0.9806 - val_loss: 0.0403 - val_accuracy: 0.9935
Epoch 9/10
78/78 - 1s - loss: 0.0739 - accuracy: 0.9846 - val_loss: 0.0983 - val_accuracy: 0.9709
Epoch 10/10
78/78 - 1s - loss: 0.0668 - accuracy: 0.9830 - val_loss: 0.0290 - val_accuracy: 0.9935
```

Figura 23:Captura de pantalla de los resultados del entrenamiento con 10 épocas y 20% de datos para validar.

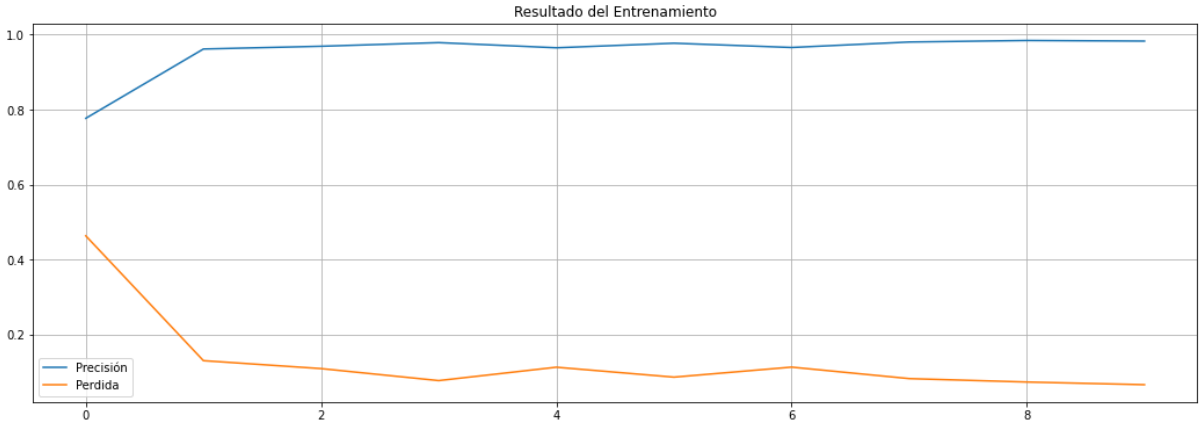

Figura 24: Gráfica de la evolución de precisión y pérdida durante el entrenamiento.

Al evaluar el modelo, luego del entrenamiento, con el conjunto de test arrojó una precisión de 99.4%.

Conclusiones de la investigación de conjuntos de datos y modelos de aprendizaje profundo

Luego de realizadas un conjunto de pruebas con el modelo de red neuronal seleccionado y los datos seleccionados, encontramos diferentes resultados para los distintos conjuntos. Por un lado se logra una baja precisión en la predicción luego del entrenamiento para los conjuntos FS1-2_LEAKING y PS1-6_LEAKING por lo que podemos concluir que las series de tiempo obtenidas a partir de los sensores de flujo y presión no explican el estado de la bomba y su fuga. En cambio se logró alta precisión en las predicciones luego del entrenamiento para los conjuntos PS1-6_VALVE y TS1-4_COOLER con lo que podemos

decir que las series de tiempo de los sensores de presión tienen correlación con el estado de la válvula y las de los sensores de temperatura tienen correlación con el estado del cooler.

Por otro lado, para las pruebas y validación del prototipo necesitaremos generar series de tiempo que alimenten la herramienta simulando que existe en realidad un equipamiento en el que los sensores envían ininterrumpidamente las lecturas realizadas. Aquí nos encontramos con una diferencia; el conjunto de datos provee series de 60 pasos de tiempo (60 lecturas) con una etiqueta de predicción al final, mientras que el prototipo deberá realizar una predicción por cada paso de tiempo.

Como conclusión de lo antes descrito necesitamos generar series de tiempo sintéticas a partir del análisis de las series de tiempo provistas por el conjunto de datos elegido para simular los distintos estados del equipamiento y la respectiva emisión de lecturas.

Al ser continua y en tiempo real la simulación que alimenta al prototipo, es importante que las series de datos del conjunto original y que son utilizadas en el entrenamiento del modelo no posean características cíclicas o estacionales que tengan relación con el periodo de 60 segundos que duran las pruebas en el conjunto de datos original, ya que al pretender crear series sintéticas se deberían reproducir estas características que no corresponden con un funcionamiento continuo del equipamiento.

De las series de datos seleccionadas en los cuatro conjuntos de datos podemos observar claramente características cíclicas en los sensores de presión y de flujo (figura 13 y figura 15) mientras que no se observan ciclos significativos en las series obtenidas de los sensores de temperatura, que reflejan más bien como el equipamiento va incrementando el valor de las lecturas con el paso de las pruebas.

Con los resultados de estos análisis respecto de la precisión del modelo y lo adecuado de las características de las series de tiempo para reproducir funcionamientos prolongados con predicciones por cada paso de tiempo seleccionamos el conjunto TS1-4_COOLER para continuar con el proyecto.

4.4. Pruebas del modelo de aprendizaje seleccionado

Según lo descrito en la documentación que acompaña al conjunto de datos elegido la plataforma hidráulica realiza pruebas con una duración de 60 segundos. Las series de datos del conjunto son las lecturas de los diferentes sensores a diferentes frecuencias que se

realizaron durante esos 60 segundos y las etiquetas corresponden al estado de los diferentes componentes monitoreados justo al terminar cada prueba.

Como el objetivo de este proyecto es lograr predecir con cierto tiempo de anticipación el estado del componente monitoreado, para poder informar y entregar en tiempo los resultados de predicción, realizamos a los conjuntos de datos seleccionados pruebas para reducir la cantidad de lecturas necesarias para que el modelo realice la predicción, sin perder precisión en el resultado. Como puede observarse en la figura 25, originalmente cada conjunto de datos posee las siguientes características:

- 2205 muestras.
- 60 pasos de tiempo en cada serie.
- 1 etiqueta con el estado del componente al finalizar la prueba de 60 segundos.

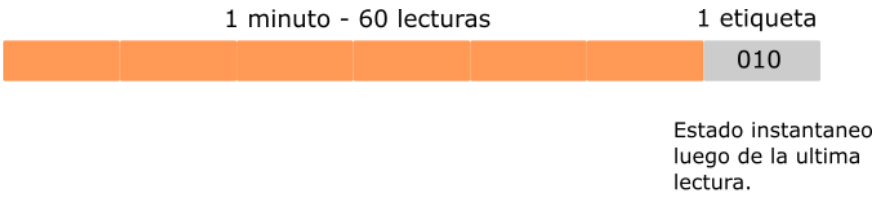

Figura 25: Una serie de tiempo en el formato original.

Las pruebas consistieron en acortar el largo de las muestras, inicialmente a las 30 primeras lecturas, luego al observar que la precisión se mantenía por encima del 95% se acortaron a las 20 primeras lecturas y como aún la precisión era superior al 95%, finalmente se dejaron solo las 10 primeras lecturas en las muestras, manteniendo el modelo una precisión superior al 98%. En la figura 26 puede observarse la representación de una muestra luego de las transformaciones descritas.

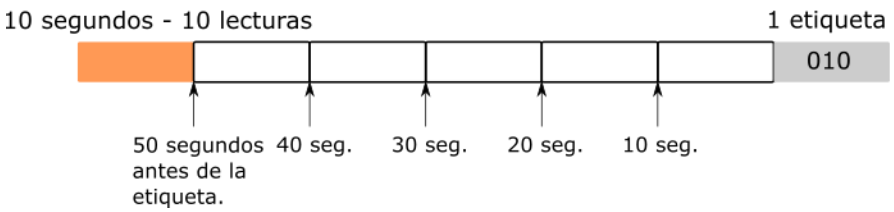

Figura 26: Una serie de tiempo en el formato reducido a 10 pasos de tiempo.

Con estas modificaciones logramos que el modelo prediga con precisión el estado que tendrá el componente 50 segundos después de la última lectura y logramos así ese espacio de tiempo entre la predicción y el momento en que el componente presente ese estado.

Las modificaciones realizadas a los conjuntos de datos repercuten en la estructura del modelo. A continuación se resumen los dos modelos aplicados en adelante para ambos conjuntos de datos.

Conjunto de Datos TS1-4_COOLER

Para el conjunto de datos **TS1-4_COOLER** la entrada está compuesta por 2205 matrices y la salida por 2205 vectores binarios. Como puede observarse en la figura 27, los atributos de cada muestra tienen la forma de una matriz de 4 x 10 donde las 4 filas representan los sensores y las columnas las 10 lecturas.

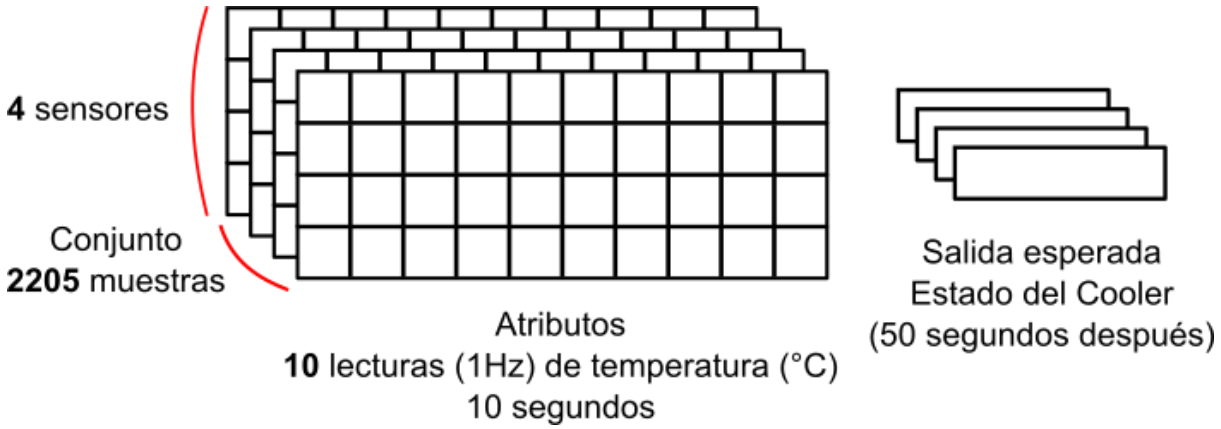

Figura 27: Entradas y salidas para el modelo TS1-4_COOLER.

La etiqueta de salida esperada, se almacena con codificación One-Hot, en la primera columna de la tabla 13 pueden observarse los tres posibles valores.

Código One-hot	Etiqueta	Estado de la válvula
[1, 0, 0]	3	Próximo a falla total.
[0, 1, 0]	20	Eficiencia reducida.
[0, 0, 1]	100	Eficiencia total.

Tabla 13: Vectores binarios de las salidas esperadas con codificación One-Hot.

Modelo de aprendizaje profundo para predicción con el conjunto de datos TS1-4_COOLER

A continuación se describen las capas de la red neuronal utilizada para el conjuntos de datos TS1-4_COOLER (se indica la línea en python que define las capas):

1) Capa convolucional 1D con 100 filtros de tamaño 6 y función de activación Relu.

```
model.add(layers.Conv1D(100, 6, activation='relu', input_shape=(10, 4)))
```

2) Capa convolucional 1D con 100 filtros de tamaño 3 y función de activación Relu.

```
model.add(layers.Conv1D(100, 3, activation='relu'))
```

3) Capa GlobalAveragePooling1D.

```
model.add(layers.GlobalAveragePooling1D())
```

4) Capa Dropout con una tasa de 0,5.

```
model.add(layers.Dropout(0.5))
```

5) Capa de salida, completamente conectada, con tres unidades de salida, una por cada clase y función de activación Softmax.

```
model.add(layers.Dense(3, activation='softmax'))
```

Modelo para el conjunto TS1-4_COOLER (input: 10 pasos, 4 series)(out: 3 clases)	
Capa	**Cantidad de Parámetros Entrenables**
1er. convolucional 1D	2500
2er. convolucional 1D	30100
Salida completamente conectada	303
TOTAL PARÁMETROS ENTRENABLES	32903

Tabla 14: Cantidad de parámetros que entrena el modelo por capa y totales.

Para el entrenamiento del modelo se utilizó entropía cruzada como función de pérdida y Adam como algoritmo de optimización.

Resultados del entrenamiento del modelo con el conjuntos de datos TS1-4_COOLER

Mediante un script de Python[16] se realizó el entrenamiento del modelo, luego de los cambios realizados se continuó utilizando 10 épocas alcanzando una precisión del 99,3% con el conjunto de validación (Figura 28). Luego graficamos la evolución de los valores de pérdida y precisión durante el proceso de entrenamiento (Figura 29).

[16] *https://colab.research.google.com/drive/1jZc0lBBG9iOeO0y-TH9aFclQwcQdTJIE?usp=sharing*

```
fit_history = model.fit(X_train, y_train, batch_size=16, epochs=10, validation_split=0.2, verbose=2)
```

```
Epoch 1/10
78/78 - 1s - loss: 0.7422 - accuracy: 0.6734 - val_loss: 0.3679 - val_accuracy: 0.8220
Epoch 2/10
78/78 - 0s - loss: 0.2486 - accuracy: 0.9408 - val_loss: 0.1398 - val_accuracy: 0.9773
Epoch 3/10
78/78 - 0s - loss: 0.1386 - accuracy: 0.9716 - val_loss: 0.0723 - val_accuracy: 0.9838
Epoch 4/10
78/78 - 0s - loss: 0.1057 - accuracy: 0.9806 - val_loss: 0.0605 - val_accuracy: 0.9871
Epoch 5/10
78/78 - 0s - loss: 0.1057 - accuracy: 0.9806 - val_loss: 0.0561 - val_accuracy: 0.9871
Epoch 6/10
78/78 - 0s - loss: 0.0943 - accuracy: 0.9838 - val_loss: 0.0616 - val_accuracy: 0.9838
Epoch 7/10
78/78 - 0s - loss: 0.1085 - accuracy: 0.9733 - val_loss: 0.0548 - val_accuracy: 0.9871
Epoch 8/10
78/78 - 0s - loss: 0.0997 - accuracy: 0.9797 - val_loss: 0.0503 - val_accuracy: 0.9871
Epoch 9/10
78/78 - 0s - loss: 0.0822 - accuracy: 0.9814 - val_loss: 0.0485 - val_accuracy: 0.9903
Epoch 10/10
78/78 - 0s - loss: 0.0868 - accuracy: 0.9846 - val_loss: 0.0430 - val_accuracy: 0.9903
```

Figura 28: Captura de pantalla de los resultados del entrenamiento con 10 épocas y 20% de datos para validar.

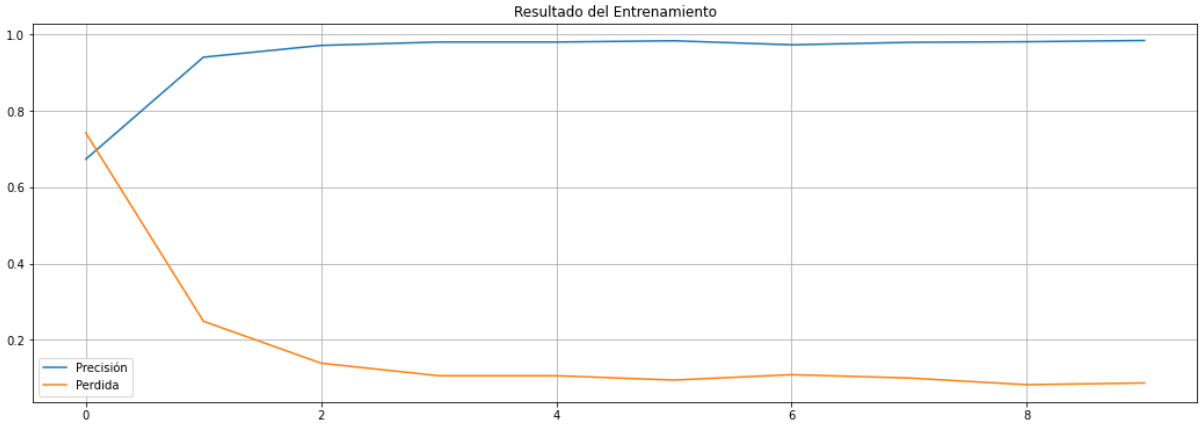

Figura 29: Gráfica de la evolución de precisión y pérdida durante el entrenamiento.

4.5. Optimización y ajustes del modelo

Si bien la precisión alcanzada por el modelo es satisfactoria y permite seguir adelante con las siguientes etapas del proyecto, es necesario analizar el modelo de red neuronal y optimizar su estructura para asegurarnos de no estarlo sobredimensionando, lo que llevaría a un desaprovechamiento del poder de cómputo.

Para optimizar el modelo de aprendizaje profundo seleccionado es necesario ajustar de alguna manera los hiperparámetros del mismo, para realizar este ajuste utilizamos el concepto de “Optimalidad de Pareto” introducido en la subsección 2.6.

En principio este método se trata de un procedimiento experimental que busca optimizar múltiples objetivos, para nuestro caso estos objetivos serán el tiempo que demora el modelo en realizar una predicción y la precisión alcanzada durante el proceso de entrenamiento.

Los hiperparámetros que se desean optimizar son:

- Tamaño de la ventana de la primera capa convolucional.
- Cantidad de filtros de la primera capa convolucional.
- Tamaño de la ventana de la segunda capa convolucional.
- Cantidad de filtros de la segunda capa convolucional.
- Tasa de la capa Dropout.

Hiperparámetro	**Valores de las pruebas**
Tamaño de Ventana de la primera capa convolucional.	`[3, 4, 5, 6]`
Cantidad de filtros de la primera capa convolucional.	`[60, 80, 100]`
Tamaño de la ventana de la segunda capa convolucional.	`[2, 3, 4]`
Cantidad de filtros de la segunda capa convolucional.	`[60, 80, 100]`
Tasa de la capa Dropout.	`[.4, .5, .6]`

Tabla 15: Hiperparámetros que varían en las pruebas y los valores que toman .

De la combinación de los valores propuestos para las pruebas (Tabla 15) resultan 324 experimentos que se evalúan según el procedimiento para identificar la Frontera de Pareto y así poder seleccionar una configuración de parámetros dependiendo de nuestras prioridades con respecto a los objetivos evaluados.

Resultados de la optimización del modelo TS1-4_COOLER

El script generado[17] para llevar adelante el proceso de optimización nos permite visualizar en una gráfica (Figura 30) cuáles fueron los experimentos que mejor desempeño

[17] *https://colab.research.google.com/drive/1eXDDLQ7V5jUezEccr1YDAlCvdUlsVb69?usp=sharing*

tuvieron y se ubicaron en la “frontera de Pareto”, de esta forma pudimos seleccionar el que a nuestro criterio era óptimo.

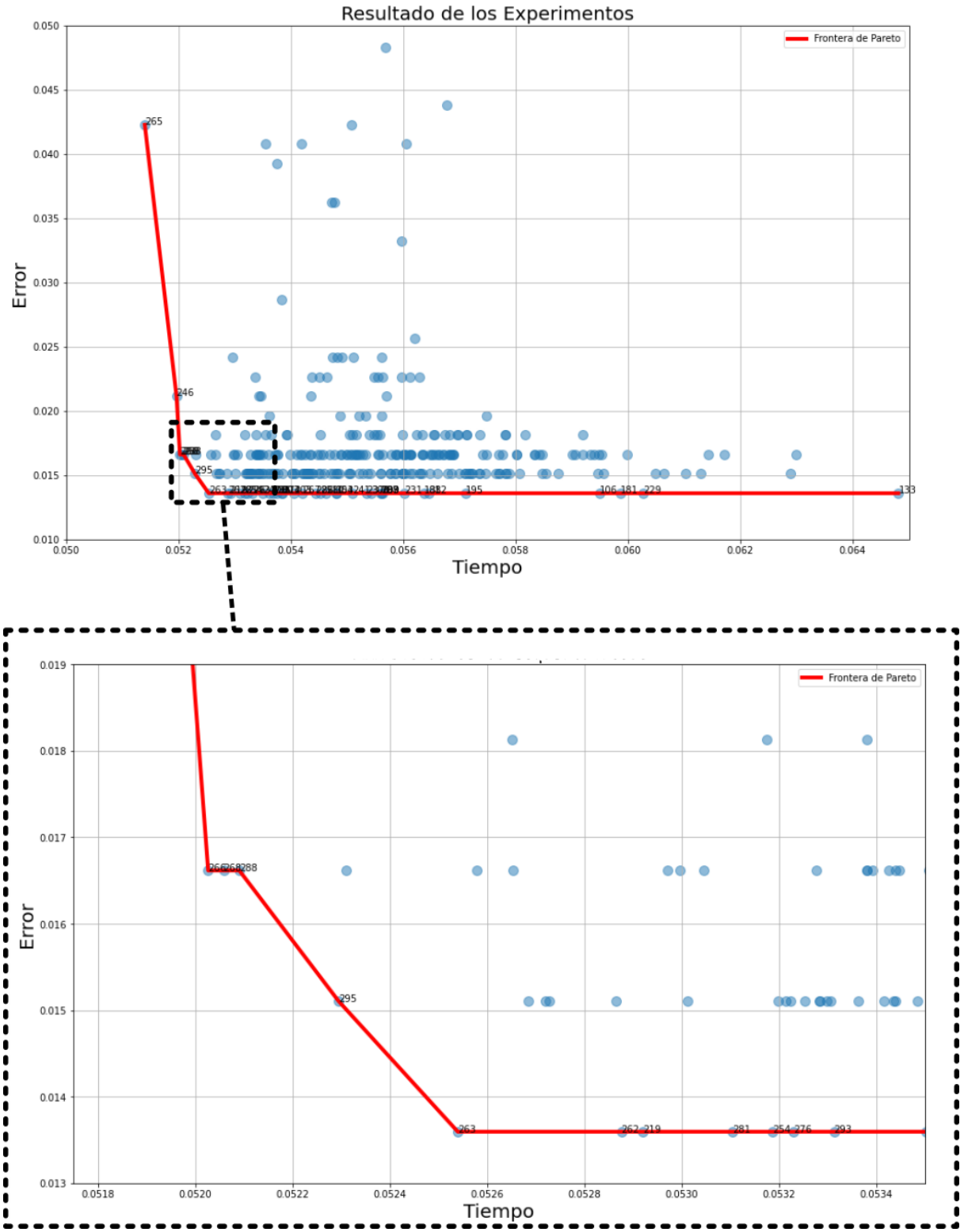

Figura 30: Captura de los resultados de la optimización y resultado seleccionado.

También, luego de realizar el procedimiento de optimización de Pareto obtenemos una lista de los experimentos que se encuentran en la frontera, donde podemos visualizar los parámetros de las mejores soluciones para esta optimización multiobjetivo (Figura 31).

```
Valores de los hiperparametros en los experimentos de la Frontera de Pareto
Experimento:  265  Hiperparametros:  [ 6.  60.   4.  80.   0.5]
Experimento:  246  Hiperparametros:  [ 6.  60.   2.  80.   0.4]
Experimento:  266  Hiperparametros:  [ 6.  60.   4.  80.   0.6]
Experimento:  268  Hiperparametros:  [  6.   60.    4.  100.    0.5]
Experimento:  288  Hiperparametros:  [ 6.  80.   4.  60.   0.4]
Experimento:  295  Hiperparametros:  [  6.   80.    4.  100.    0.5]
Experimento:  263  Hiperparametros:  [ 6.  60.   4.  60.   0.6]
Experimento:  262  Hiperparametros:  [ 6.  60.   4.  60.   0.5]
Experimento:  219  Hiperparametros:  [  5.  100.    2.   80.    0.4]
Experimento:  281  Hiperparametros:  [ 6.  80.   3.  60.   0.6]
Experimento:  254  Hiperparametros:  [ 6.  60.   3.  60.   0.6]
Experimento:  276  Hiperparametros:  [  6.   80.    2.  100.    0.4]
Experimento:  293  Hiperparametros:  [ 6.  80.   4.  80.   0.6]
Experimento:  247  Hiperparametros:  [ 6.  60.   2.  80.   0.5]
Experimento:  308  Hiperparametros:  [  6.  100.    3.   60.    0.6]
Experimento:  285  Hiperparametros:  [  6.   80.    3.  100.    0.4]
Experimento:  301  Hiperparametros:  [  6.  100.    2.   80.    0.5]
Experimento:  303  Hiperparametros:  [  6.  100.    2.  100.    0.4]
Experimento:  274  Hiperparametros:  [ 6.  80.   2.  80.   0.5]
Experimento:  203  Hiperparametros:  [ 5.  80.   3.  80.   0.6]
Experimento:  267  Hiperparametros:  [  6.   60.    4.  100.    0.4]
Experimento:  286  Hiperparametros:  [  6.   80.    3.  100.    0.5]
Experimento:  258  Hiperparametros:  [  6.   60.    3.  100.    0.4]
Experimento:  210  Hiperparametros:  [ 5.  80.   4.  80.   0.4]
Experimento:  304  Hiperparametros:  [  6.  100.    2.  100.    0.5]
Experimento:  153  Hiperparametros:  [  4.  100.    4.   60.    0.4]
Experimento:  241  Hiperparametros:  [  5.  100.    4.  100.    0.5]
Experimento:  237  Hiperparametros:  [  5.  100.    4.   80.    0.4]
Experimento:  306  Hiperparametros:  [  6.  100.    3.   60.    0.4]
Experimento:  287  Hiperparametros:  [  6.   80.    3.  100.    0.6]
Experimento:  198  Hiperparametros:  [ 5.  80.   3.  60.   0.4]
Experimento:  309  Hiperparametros:  [  6.  100.    3.   80.    0.4]
Experimento:  231  Hiperparametros:  [  5.  100.    3.  100.    0.4]
Experimento:  188  Hiperparametros:  [  5.   60.    4.  100.    0.6]
Experimento:  312  Hiperparametros:  [  6.  100.    3.  100.    0.4]
Experimento:  195  Hiperparametros:  [  5.   80.    2.  100.    0.4]
Experimento:  106  Hiperparametros:  [  4.   60.    4.  100.    0.5]
Experimento:  181  Hiperparametros:  [ 5.  60.   4.  60.   0.5]
Experimento:  229  Hiperparametros:  [  5.  100.    3.   80.    0.5]
Experimento:  133  Hiperparametros:  [  4.   80.    4.  100.    0.5]
```

Figura 31: Captura de los resultados de la optimización.

Debido a las pequeñas diferencias de tiempo en las predicciones procedemos a seleccionar dentro del grupo de experimentos con mejor precisión, el de menor tiempo de

predicción, entonces, el experimento seleccionado es el 263 con los siguientes hiperparámetros de prueba que se visualizan en la tabla 16.

Hiperparámetro	**Valor**
Tamaño de Ventana de la primera capa convolucional.	6
Cantidad de filtros de la primera capa convolucional.	60
Tamaño de la ventana de la segunda capa convolucional.	4
Cantidad de filtros de la segunda capa convolucional.	60
Tasa de la capa Dropout.	.6

Tabla 16: Parámetros del experimento seleccionado de la optimización.

Entrenamiento final del modelo y guardado de los datos

Con los resultados de la optimización procedemos a entrenar el modelo y guardar luego, tanto la estructura del modelo como también los valores de los parámetros entrenados, esto es necesario para luego a partir de estos archivos cargar el modelo y los parámetros al prototipo.

A partir de un script de Python[18] se almacena el modelo de aprendizaje profundo en un archivo JSON y los parámetros entrenados en un archivo H5(Hierarchical Data Format). De esta forma con Keras podemos leer estos archivos y utilizar el modelo entrenado en el prototipo.

[18] *https://colab.research.google.com/drive/10luuSlUFsuA-VmsG51moqWZlMx8STYNN?usp=sharing*

4.6. Diseño y desarrollo del prototipo de herramienta de software

En esta subsección se enumeran los objetivos y requerimientos del prototipo, y se explica el diseño propuesto para cumplir con estos y el desarrollo implementado de acuerdo a este diseño. También se detalla la creación de series de datos sintéticas requeridas por el prototipo diseñado para simular las lecturas de instrumentos.

Requerimientos del prototipo de herramienta de software

Uno de los objetivos de este trabajo es el desarrollo de un prototipo de herramienta de software, que pretende simular el proceso de recepción de datos en tiempo real de un determinado dispositivo y predecir, a partir de estos datos y los recibidos anteriormente, el estado de un determinado componente. El prototipo debe cumplir con los siguientes requerimientos:

- Un proceso que simule la emisión de lecturas con una cierta frecuencia que se ajustará de acuerdo a los parámetros reales de los equipos estudiados.

- Un proceso que capture las lecturas emitidas por el proceso simulador de lecturas y a partir de estas lecturas pueda predecir y emitir un mensaje con el estado predicho del componente o equipamiento monitoreado en un cierto lapso de tiempo futuro.

- El proceso predictor debe tener la capacidad de leer distintos modelos entrenados para distintos tipos de componentes y series de tiempo.

- Debido a que se trata de dos procesos asíncronos, por un lado el emisor de lecturas y por el otro en predictor de estado, es necesario la existencia de un repositorio al que ambos procesos tengan acceso.

Modelo propuesto del prototipo de herramienta de software

A partir del objetivo específico del proyecto, los requerimientos planteados para el prototipo, y los modelos y artefactos propuestos para el tratamiento de los datos y el uso del modelo de aprendizaje, se propone el siguiente modelo que representa tanto los dos procesos de simulación y predicción, como también el repositorio que comparten.

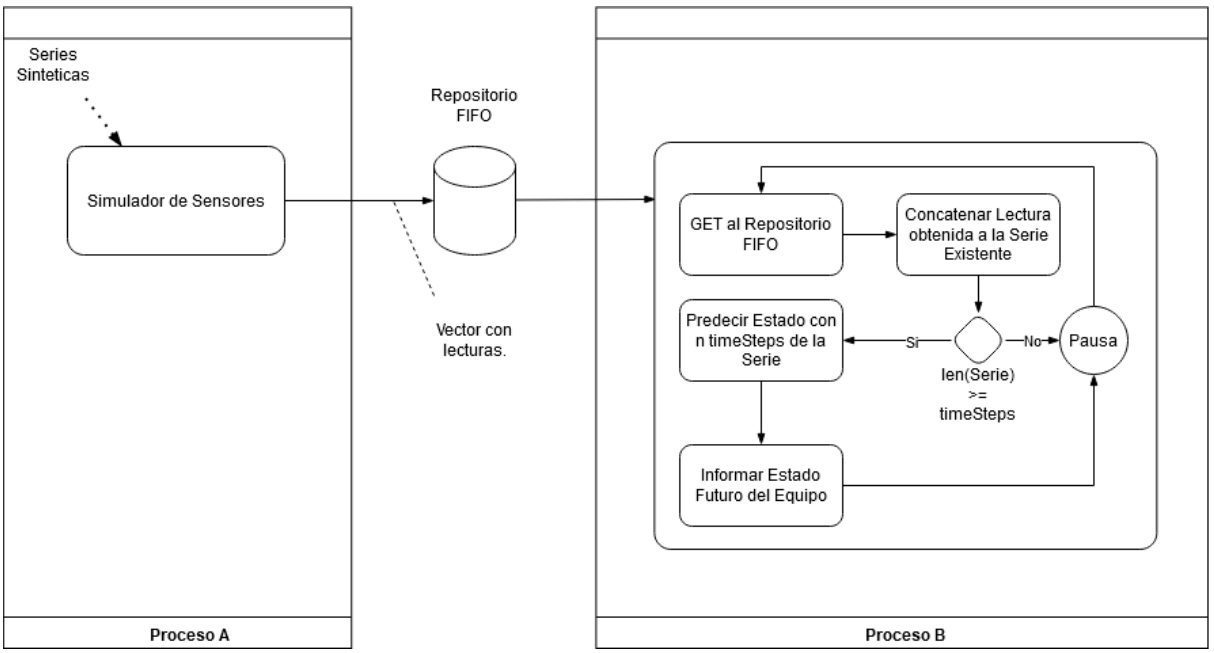

Figura 32: Modelo propuesto del prototipo.

Como se puede observar en la figura 32, los procesos A y B son independientes, por lo que se deben implementar mediante programación concurrente en dos hilos distintos de ejecución. El repositorio que comunica ambos procesos es una cola (FIFO) para mantener el orden de las lecturas en caso de demoras en el Proceso B que causen acumulación de datos en el repositorio. El repositorio debe tener implementado un mecanismo de exclusión mutua (MUTEX) para evitar conflictos en caso de acceso simultáneo de ambos procesos al repositorio.

Generación de series de datos para alimentar el proceso simulador (A)

Las series de datos provistas por el conjunto de datos pertenecen a muestras de un minuto de duración. Luego de las modificaciones y ajustes realizados en la subsección 4.4, su duración se reduce a diez lecturas correspondientes a los últimos diez datos leídos.

Como el objetivo de este trabajo es un prototipo que prediga el estado en tiempo real, nos encontramos con la necesidad de crear series de tiempo sintéticas que emulen el comportamiento de un equipo en distintas situaciones para poder, al final, validar el funcionamiento de nuestro prototipo.

Estas series sintéticas se construyen mediante un script en Python[19], utilizando el análisis del conjunto de datos TS1-4_COOLER del cual se pueden identificar las características principales de estas series de datos. Como resultado, se obtienen series sintéticas de 150 segundos, emulando las características del conjunto de datos de entrada para distintas etiquetas de estado del componente monitoreado.

Desarrollo del prototipo de herramienta de software

En función del modelo indicado en la Figura 32, teniendo en cuenta que el prototipo deberá soportar distintos conjuntos de datos y que se necesitarán dos procesos asíncronos que compartan un repositorio para intercambiar los datos, se decide implementar el prototipo en el lenguaje Python. Se utiliza para tal fin un entorno local, con el objetivo de garantizar la estabilidad y control de los procesos concurrentes.

En la Figura 33 se presenta el diagrama de flujo del prototipo implementado. En primer lugar se leen las series de datos que alimentan el proceso simulador. Luego, se leen los datos necesarios para el proceso predictor (modelo, parámetros entrenados y parámetros de normalización). Una vez obtenidos todos los datos necesarios para ambos procesos, se inician operando de forma totalmente asíncrona. Mientras el proceso simulador se encarga de depositar en el repositorio de intercambio FIFO un vector de lecturas por cada paso de tiempo, el proceso predictor está constantemente observando el repositorio. Si este encuentra un nuevo vector de lecturas, lo concatena a los ya recibidos, toma la cantidad necesaria de últimas lecturas (si ya dispone de esa cantidad) y realiza la predicción.

Para finalizar el proceso predictor, entrega el resultado de la predicción que puede ser en una gráfica, en texto por pantalla y/o guardada en un archivo con el historial de predicciones.

[19] *https://colab.research.google.com/drive/1kUcsS1xvoobn6ZXK7qe5jsb8mzq0S61b?usp=sharing*

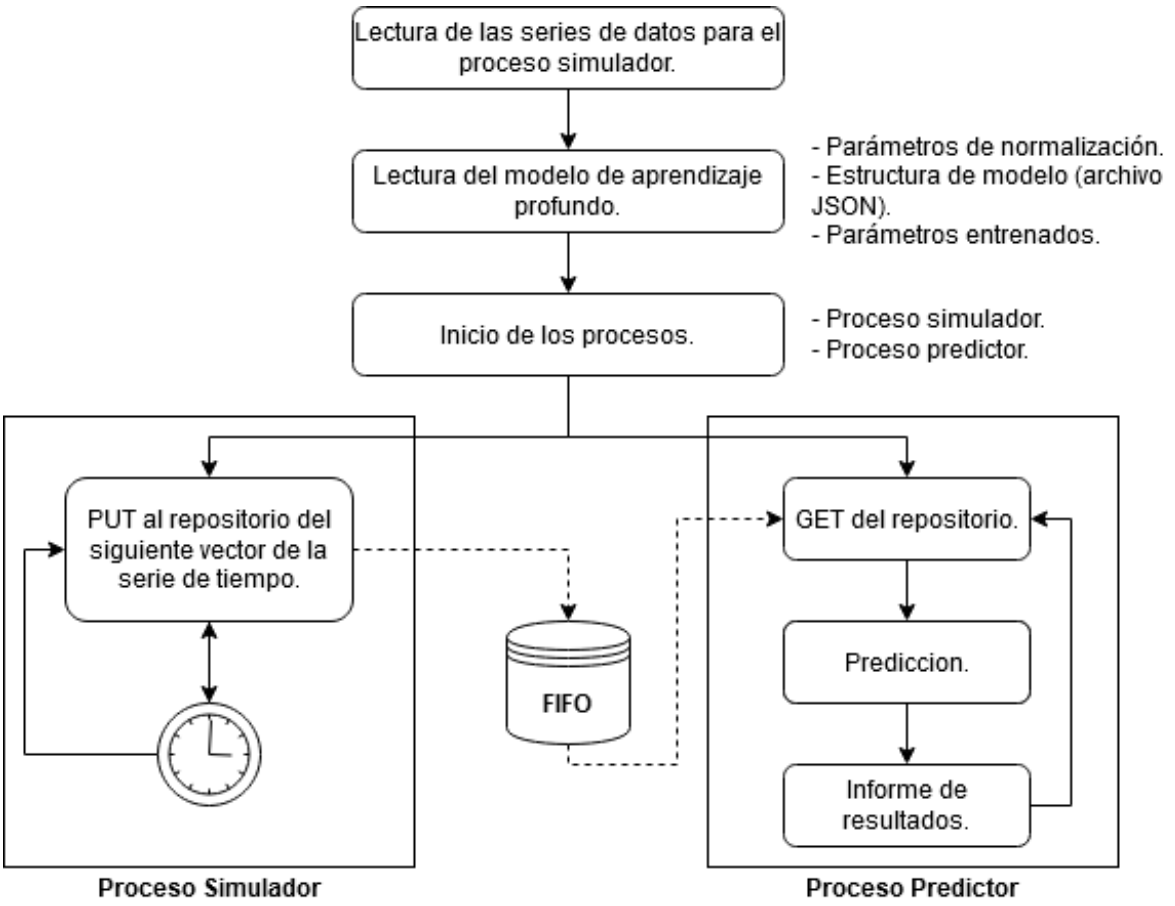

Figura 33: Diagrama de flujo del prototipo.

4.7. Pruebas y validación del prototipo

Para las pruebas del prototipo con el conjuntos de datos TS1-4_COOLER analizado en la subsección 4.2 se utilizó el modelo probado en la subsección 4.4 y optimizado mediante el procedimiento de Optimalidad de Pareto en la subsección 4.5 que obtuvo una precisión superior al 98% durante el entrenamiento.

Pruebas con series de datos sintéticas

El proceso Simulador se alimentó para las pruebas con series de datos sintéticas (se pueden observar en la Figura 34) mediante un script de Python de acuerdo al análisis que se realizó de las características de las series provistas por los sensores y del comportamiento de la etiqueta de salida que muestra el estado del COOLER. En la tabla 17 se visualizan los valores promedio de las lecturas en las muestras de cada sensor para los distintos estados del componente COOLER. También se puede observar un aumento de los valores medios reportados por los sensores, de derecha a izquierda, a medida que el estado del componente

pasa de “Buen Estado” a “Medio” y luego a “Crítico”. De acuerdo a estas observaciones, las series de tiempo tienen tendencia ascendente en los valores con la incorporación de un pequeño ruido, simulando las lecturas de los sensores en un equipo que va incrementando su temperatura.

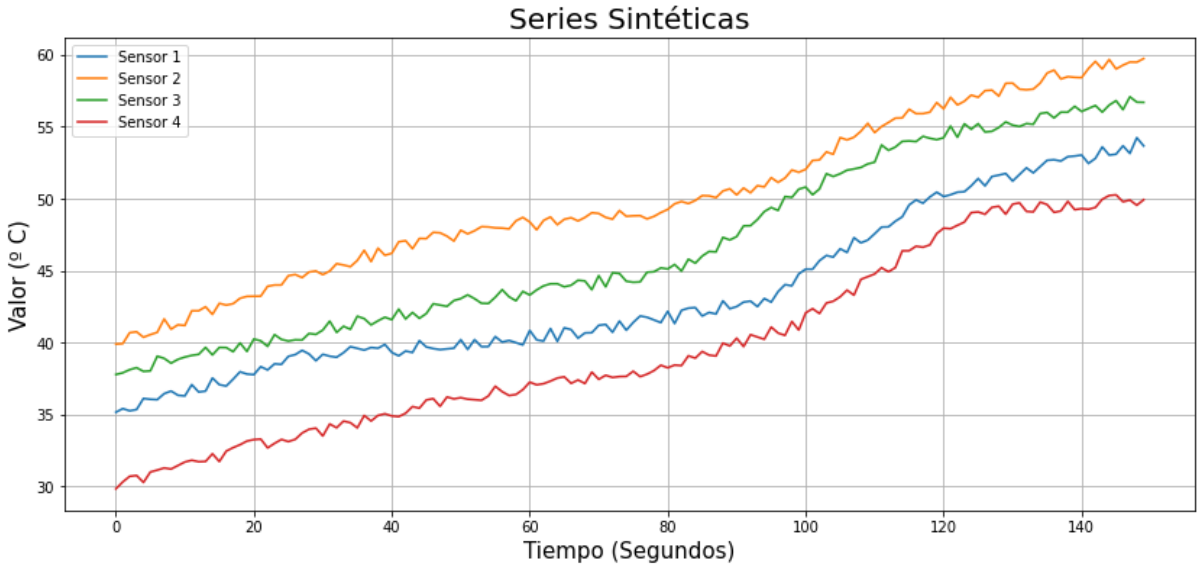

Figura 34: Series de tiempo utilizadas en las pruebas con el modelo entrenado para el conjunto TS1-4_COOLER.

	Critico	Medio	Buen Estado
0	54.783402	45.244096	35.757489
1	59.141513	50.382846	41.516762
2	56.520999	47.630990	38.782133
3	50.309148	40.912500	30.910863

Tabla 17: Valores medios reportados por los sensores TS1-4 para cada etiqueta de salida

Resultados de las pruebas realizadas con series de datos sintéticas

El prototipo[20] muestra por consola las salidas de ambos procesos para poder seguir su funcionamiento y tener registro de toda la corrida a fin de verificar que todo se ejecuta según lo esperado. También permite visualizar en una ventana la gráfica con la evolución de la serie que está alimentando al proceso simulador.

[20] *https://github.com/maurojp/Py_pred_sim/blob/master/h_p_s_1.py*

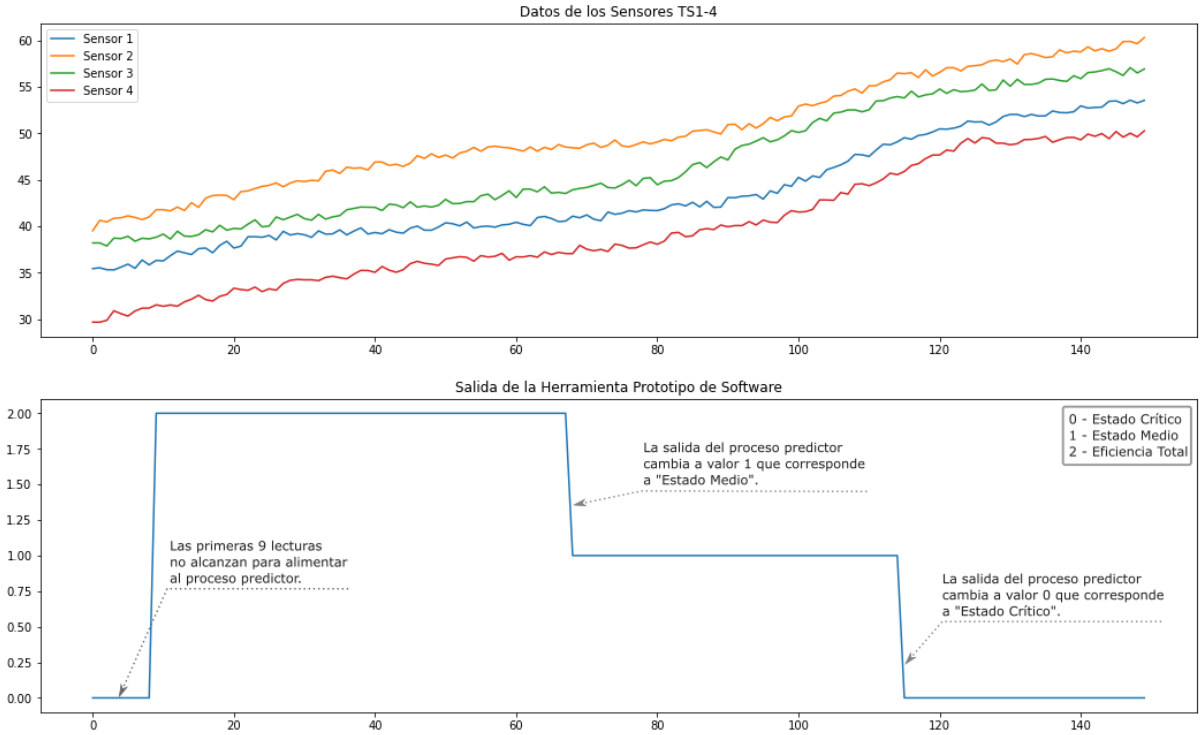

Figura 35: Gráfica de la serie de tiempo y la variable de salida del prototipo.

En las gráficas de la figura 35 se puede observar un buen comportamiento del proceso predictor. En principio no realiza predicciones hasta tener 10 lecturas almacenadas, ya que esta es una necesidad del modelo convolucional. Al comienzo las predicciones tienen valor 2, que corresponde a la etiqueta “Eficiencia Total”. Luego, al incrementarse los valores de temperatura podemos apreciar que, a partir de la predicción 69 la salida cambia a valor 1, que corresponde a la etiqueta “Estado Medio”. Y a partir de la predicción 115, la salida tiene valor 0, que corresponde a la etiqueta “Estado Crítico”. Entonces, la salida predicha varía para el estado del COOLER a medida que los valores reportados por los sensores aumentan.

Conjuntos de datos de la industria para prueba del prototipo

Si bien se realizaron pruebas y se obtuvieron resultados favorables con series de datos sintéticas, a partir de un conjunto de datos provisto por la industria se entrenó un modelo y ajustó el prototipo para ver el comportamiento con datos del mundo real. Las características de este nuevo conjunto de datos difieren mucho del conjunto TS1-4_COOLER, lo que plantea un desafío en la prueba del prototipo en un contexto real.

En primer lugar se realizó un análisis exploratorio de datos de este conjunto anonimizado por restricción de la industria que lo proveyo. El conjunto de datos consiste de

una serie de 912 elementos, cada uno de los cuales contiene la lectura entregada por 119 sensores de flujo cada media hora y una columna denominada “TARGET” que es la etiqueta de salida.

Como datos de la etiqueta solo se aclara que esta identifica una falla cuando su valor decrece significativamente. A fin de reflejar un estado binario que indique si se presenta una falla o no, se agregó al conjunto de datos una columna denominada “LABEL(T+1)”. Ésta contiene un valor 0 si el valor de la columna “TARGET” es mayor a 3.00 en el paso de tiempo siguiente (media hora después). Caso contrario, el valor de “LABEL(T+1)” es 1, indicando un valor bajo de la variable TARGET, significando una posible falla. Cabe aclarar que el valor 3 como umbral para el cambio de estado de la variable “LABEL(T+1)” se seleccionó de forma arbitraria, analizando visualmente el comportamiento de los datos .

En la figura 36 podemos observar las gráficas de la evolución en el espacio de muestras de algunas de las 119 variables seleccionadas al azar que componen los atributos del conjunto de datos. Claramente se pueden apreciar diferentes comportamientos en los valores que toman estas variables con el paso del tiempo. También se puede observar, en la subgráfica de la variable “LABEL(T+1)”, que si la utilizamos como salida esperada para entrenar un modelo tendríamos un desbalance importante en la cantidad de casos negativos y positivos. Esto se evidencia numéricamente en la Tabla 18.

Distribución para la salida “LABEL(T+1)”	
Estado	**Cantidad de muestras**
Buen estado. (Valor 0)	859
Estado crítico. (Valor 1)	54

Tabla 18: Distribución de casos positivos y negativos en el conjunto de datos provisto por la industria

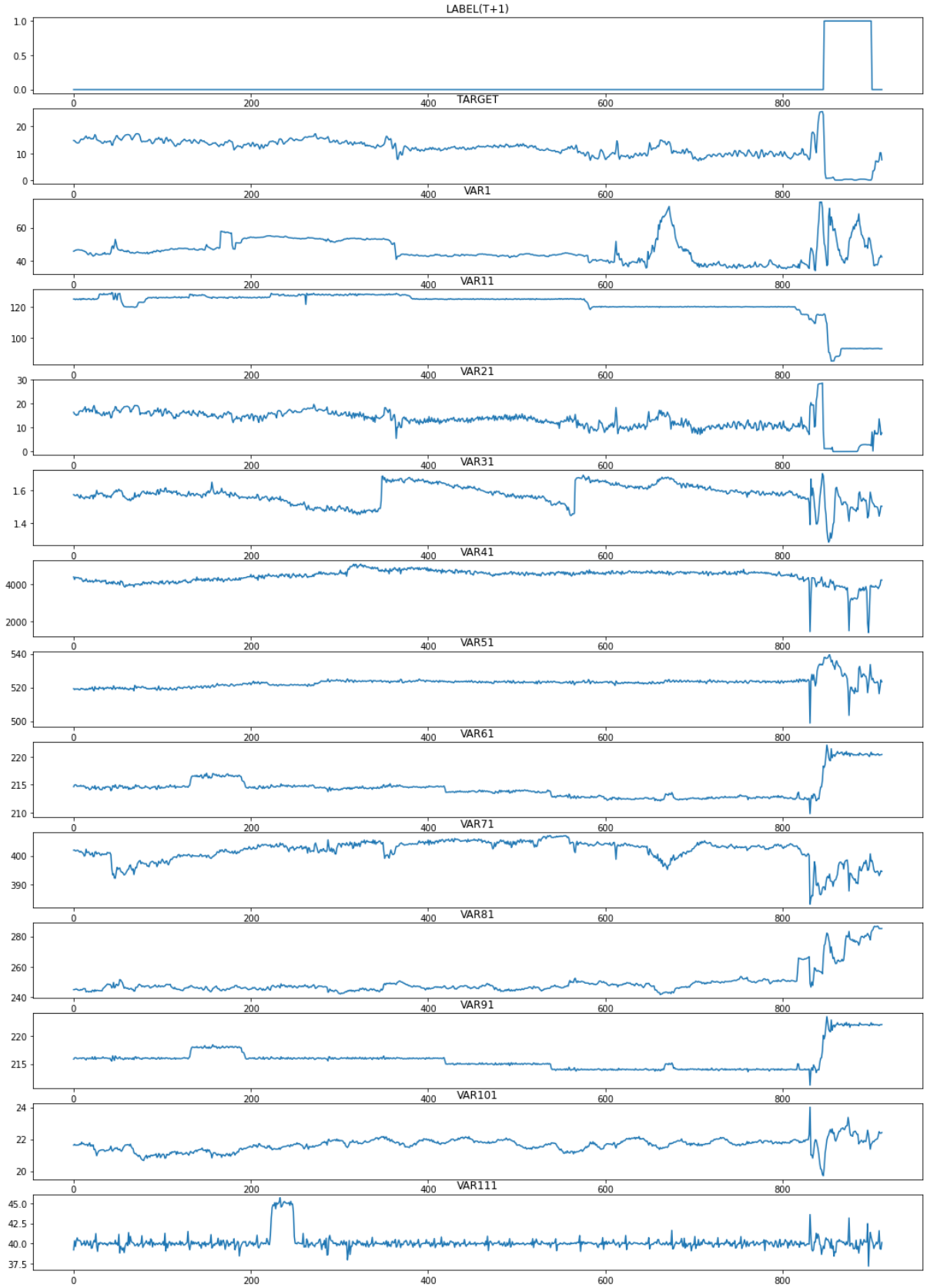

Figura 36: Gráficas de algunas de las 119 variables, la variable "TARGET" y el valor "LABEL(T+1)".

Para solucionar el desbalance decidimos probar una estrategia básica de ataque de este problema que consta de reducir la cantidad de muestras de la clase mayoritaria a la misma cantidad de muestras de la clase minoritaria. Para este caso entonces tomamos solo desde la muestra 793 y hasta la muestra 900, de esta forma el conjunto de datos queda balanceado con 54 casos negativos y 54 positivos.

Modelo seleccionado para el conjunto de datos de la industria

El conjunto de datos provisto consta de una sola serie de datos, por lo que no creímos conveniente el uso de una red neuronal convolucional debido a la escasa cantidad de datos para un modelo de este tipo. Por lo antes expuesto se implementó un modelo de red neuronal Perceptrón Multicapa explicado en la subsección 2.4. El dataset se compone de una entrada con las lecturas de los 119 sensores y una salida que es el valor del estado media hora después. Para optimizar el modelo se utilizó nuevamente el procedimiento de Optimalidad de Pareto realizando un conjunto de experimentos con variaciones en distintos hiperparámetros.

Hiperparámetro	**Valores de las pruebas**
Cantidad de unidades de la primer capa oculta	`[100, 120, 140]`
Cantidad de unidades de la segunda capa oculta	`[40, 50, 60, 80]`
Cantidad de unidades de la tercera capa oculta	`[0, 10 20]`

Tabla 19: Hiperparámetros que varían en las pruebas y los valores que toman .

Como se puede ver en la tabla 19, los hiperparámetros que se varían durante las pruebas de Optimalidad de Pareto son las cantidades de unidades de la primera, segunda y tercera capas ocultas y la cantidad de capas ocultas. Esto ocurre ya que para el valor 0 en la tercera capa oculta, el script realizado en Python omite la capa, resultando el modelo con dos capas ocultas. Con esta configuración de pruebas se realizan en total, combinando los valores de los hiperparámetros, 36 experimentos.

Al igual que con el conjunto de datos anterior, se confeccionó un script en Python[21] para llevar adelante el proceso de optimización. En la gráfica de la figura 37 podemos observar sobre la frontera resaltada en color rojo los experimentos que mejor desempeño tuvieron. En esta se evidencian pequeñas variaciones en el tiempo de predicción entre los

[21] *https://colab.research.google.com/drive/1XefF_b3-wJgzXuBi-9oGjJvyui0ffWvo?usp=sharing*

modelos, que para el caso de nuestra aplicación son despreciables. Por esto, se elige el experimento con menor tiempo de predicción del grupo de los que obtuvieron menor error, en nuestro caso es el experimento 27. Como se puede observar en la tabla 20, el modelo seleccionado tiene dos capas ocultas con 140 y 50 unidades respectivamente.

Resultados de la optimización del modelo para datos de la industria

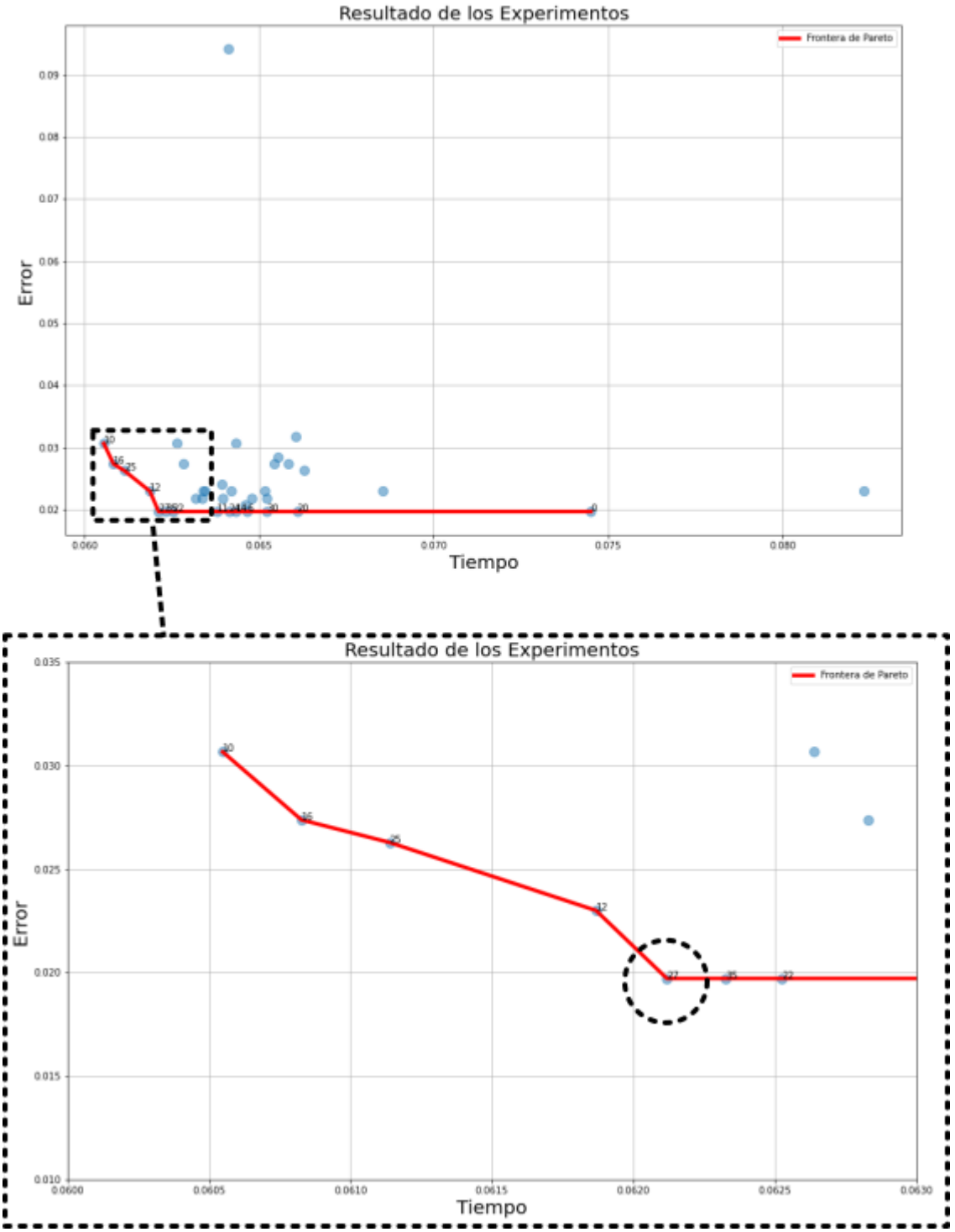

Figura 37: Captura de los resultados de la optimización y resultado seleccionado.

El script nos permitió observar una lista de los experimentos que se encuentran en la frontera, donde pudimos visualizar los parámetros de las mejores soluciones (Figura 38).

```
Valores de los hiperparametros en los experimentos de la Frontera de Pareto
Experimento:  10  Hiperparametros:  [100.  80.  10.]
Experimento:  16  Hiperparametros:  [120.  50.  10.]
Experimento:  25  Hiperparametros:  [140.  40.  10.]
Experimento:  12  Hiperparametros:  [120.  40.   0.]
Experimento:  27  Hiperparametros:  [140.  50.   0.]
Experimento:  35  Hiperparametros:  [140.  80.  20.]
Experimento:  22  Hiperparametros:  [120.  80.  10.]
Experimento:  11  Hiperparametros:  [100.  80.  20.]
Experimento:  24  Hiperparametros:  [140.  40.   0.]
Experimento:  14  Hiperparametros:  [120.  40.  20.]
Experimento:  6  Hiperparametros:  [100.  60.   0.]
Experimento:  30  Hiperparametros:  [140.  60.   0.]
Experimento:  20  Hiperparametros:  [120.  60.  20.]
Experimento:  0  Hiperparametros:  [100.  40.   0.]
```

Figura 38: Captura de los resultados de la optimización.

Hiperparámetro	**Valor**
Cantidad de unidades de la primer capa oculta	140
Cantidad de unidades de la segunda capa oculta	50
Cantidad de unidades de la tercera capa oculta	0

Tabla 20: Parámetros del experimento 27, seleccionado de la optimización.

Para el entrenamiento del modelo se utilizó entropía cruzada como función de pérdida y Adam como algoritmo de optimización.

Pruebas con el conjunto de datos de la industria

Con los valores de hiperparámetros obtenidos de la optimización entrenamos el modelo para guardar tanto la estructura en el archivo de formato JSON como los valores de los parámetros entrenables en el archivo de formato H5 y así luego poder utilizarlos en el prototipo.

Para el entrenamiento[22], debido a la poca cantidad de datos del conjunto, no utilizamos una partición para validación sino que las 108 muestras formaron el conjunto de entrenamiento mientras que para test se utilizó el conjunto completo de 912 muestras.

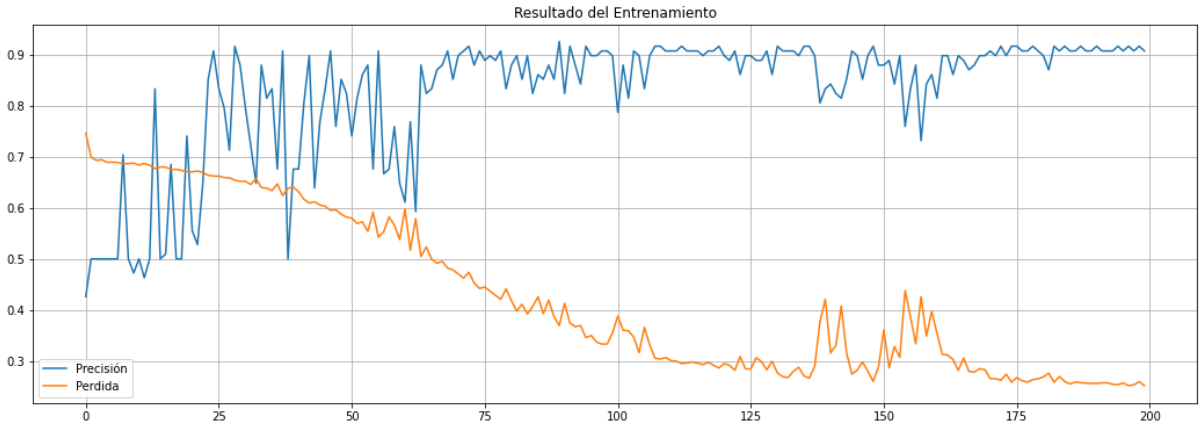

Figura 39: Evolución de precisión y pérdida durante el entrenamiento.

En la figura 39 se puede observar la evolución de la pérdida y la precisión durante del entrenamiento que resultó en 200 épocas, la precisión alcanzada con el total del conjunto de datos fue de 97.81%.

Resultados de las pruebas realizadas con el conjunto de datos de la industria

Las pruebas en este caso se realizaron con el modelo resultante de la optimización de Pareto en el proceso predictor.nte la falta de datos y la complejidad de elaborar series sintéticas con 119 variables que se comportan de forma totalmente distinta, se utilizó para alimentar el proceso simulador la serie completa de datos provista, incluso con los datos iniciales que no se utilizaron en el entrenamiento. De esta forma se obtuvo una serie de 912 pasos de tiempo, una longitud suficiente para las pruebas en el prototipo[23].

Los resultados fueron muy satisfactorios, más aún teniendo en cuenta lo reducido del conjunto de datos provisto. Se puede observar en la gráfica superior de la figura 40 que el valor de la variable TARGET desciende por debajo de 3 en el paso de tiempo 848, por lo que se observa en la segunda gráfica de la figura 40 que la variable LABEL(T+1) cambia su valor de 0 a 1 un paso de tiempo antes, es decir en el paso 847. Mientras podemos apreciar en la gráfica inferior de la figura 40 que el prototipo comienza a informar un estado futuro 1 qué

[22] *https://colab.research.google.com/drive/1M9f7Jd96N8sVUrjrMvq3rGINBN5xuUaA?usp=sharing*
[23] *https://github.com/maurojp/Py_pred_sim/blob/master/h_p_s_2.py*

significa “ESTADO CRÍTICO” a partir del paso de tiempo 831, es decir 16 pasos de tiempo antes de que los valores sean críticos. Teniendo en cuenta que el paso de tiempo para este conjunto de datos es de 30 minutos entre muestras, el prototipo está prediciendo una falla 8 horas antes en este caso.

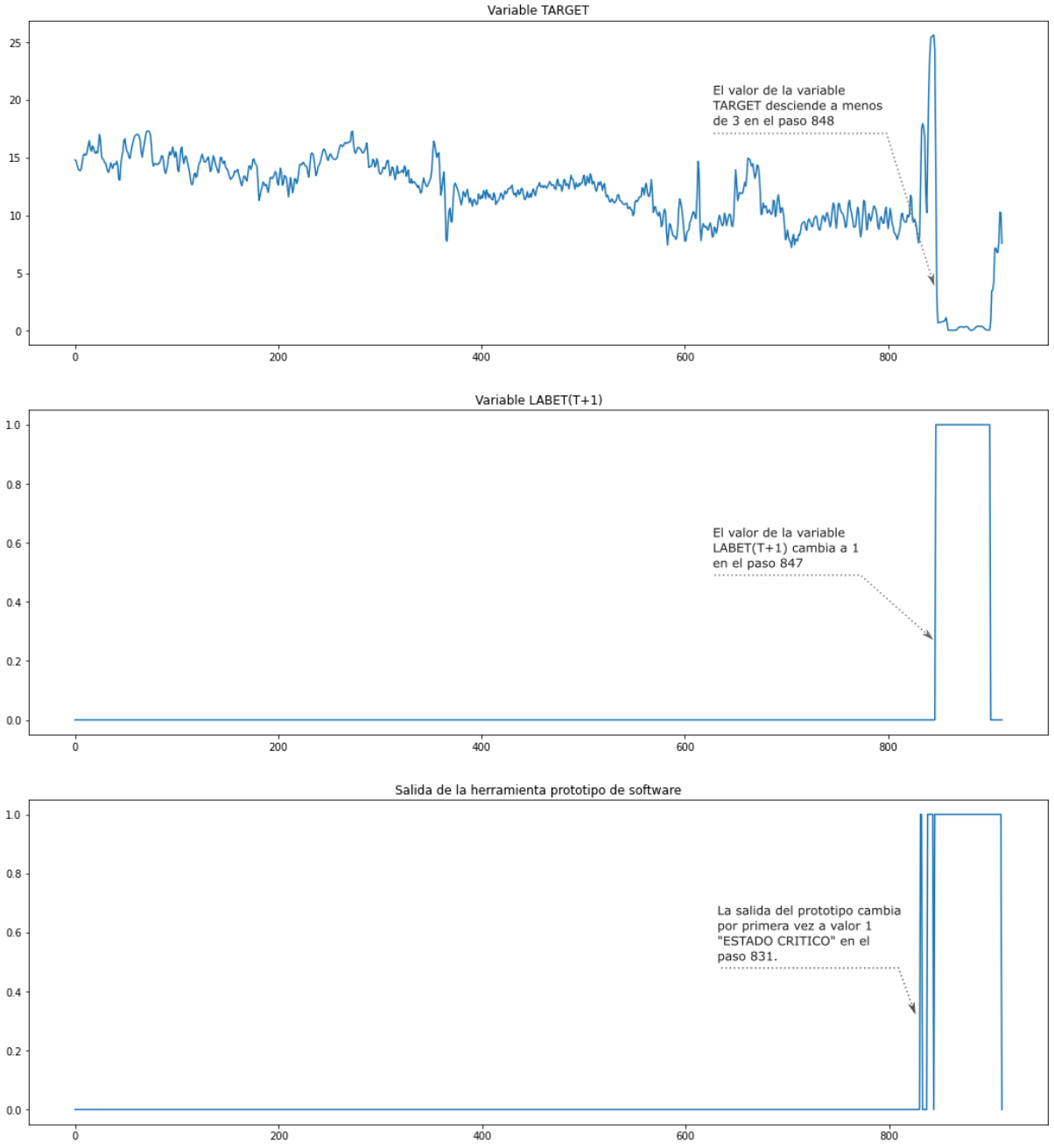

Figura 40: Gráfica de la serie provista, la variable TARGET y la salida del prototipo.

5. Conclusiones

Del desarrollo de este proyecto final de carrera, se obtiene como principal conclusión que en todo el desarrollo del mismo los integrantes hemos podido sentir como los conocimientos adquiridos durante el transcurso de la carrera han servido y se han entrelazado para poder obtener resultados, solucionar problemas, rehacer procedimientos, plantear pruebas y todo lo que hemos necesitado a lo largo de este trabajo.

En cuanto a los objetivos planteados, podemos concluir que se han cumplido satisfactoriamente, desde la investigación del "estado del arte" en la temática del proyecto, la investigación de conjuntos de datos, modelos y técnicas de aprendizaje profundo, hasta el diseño y el desarrollo del prototipo de herramienta de software que permite simular el proceso de recibir y predecir el estado futuro de un equipo o componente en tiempo real.

Desde el análisis de la literatura pudimos observar que la gran variedad de dispositivos, sensores y procesos en la industria se reflejan en una gran variedad de modelos de datos. Estas diferencias hacen necesarios distintos análisis de las series de datos en las tareas previas a la selección del modelo de aprendizaje profundo a utilizar, para determinar si serán necesarios modelos recurrentes, modelos convolucionales o alguna otra de las muchas tipologías existentes.

Se pudo estudiar en las primeras etapas que existe una variedad de modelos de aprendizaje profundo que pueden tener resultados satisfactorios si son aplicados y ajustados correctamente, de acuerdo a los conjuntos de datos con los que se desea trabajar. En este trabajo hemos podido verificar como en las pruebas iniciales funcionó muy bien un modelo convolucional y luego en las pruebas con datos de la industria, las características y escasez de los mismos hicieron necesario la implementación de un modelo perceptrón multicapa con excelentes resultados. Ambos modelos, aunque con diferente profundidad, son parte de una misma estrategia de aprendizaje automático.

La selección de los modelos de aprendizaje automático nos permitió experimentar el complejo proceso de ajustar los hiperparámetros para lograr mejores resultados en el entrenamiento y no sobredimensionar las estructuras, lo que impactaría en la necesidad de un excesivo poder de cómputo. Para este proceso fue de mucha utilidad la utilización del concepto de la Optimalidad de Pareto que nos permitió visualizar cómo los distintos ajustes impactaron en la "caja negra" que representa el modelo de aprendizaje profundo.

También fue posible proponer un prototipo de herramienta de software para tareas de mantenimiento predictivo que incorpore los resultados de la consecución de los objetivos anteriores. Así, este prototipo puede leer distintos tipos de modelos para implementarlos con distintos conjuntos de datos, y de esta manera reutilizar la herramienta reduciendo los tiempos de prueba. Esto se logró gracias a la implementación de las herramientas y artefactos tecnológicos descritos mediante el uso de archivos de intercambio .JSON para los modelos y .H5 para los parámetros entrenables. Ambos formatos de archivo pueden interpretarse por lo que podrían importarse modelos y parámetros desde otras fuentes o desarrollar herramientas propias de modelado. Además, el prototipo logró la inclusión de procesos de simulación de datos para el entrenamiento de los modelos, como así también procesos de predicción de resultados. Esto ha sido probado satisfactoriamente con datos obtenidos de la literatura como así también de entornos reales de la industria.

Trabajos futuros

Durante el desarrollo de este trabajo fueron surgiendo varias ideas para proyectos futuros. El principal trabajo futuro comprende la continuación y mejora del prototipo de herramienta de software ya que este trabajo es una versión inicial. Las próximas versiones deben comprender el agregado de interfaces que permitan una mejor experiencia del uso tanto en la configuración de lo que comprende al modelo y conjuntos de datos, como así también a la entrega de informes y gráficas de forma sencilla.

También ante la variedad de tipos de conjuntos de datos es necesario realizar pruebas en el prototipo con distintos modelos y formatos de series para continuar analizando el desempeño, y así determinar las distintas opciones de configuración que se deben agregar para el correcto funcionamiento en los distintos casos de uso.

Para los conjuntos de datos como el del primer caso donde se pueden generar series sintéticas se necesitan continuar las pruebas simulando en las series la falla de uno o varios sensores, con distintos tipos de problemas e inclusión de ruidos en las señales para observar la respuesta de la herramienta a estos eventos.

En el caso del conjunto de datos aportado por la industria, se pueden realizar modificaciones sobre los atributos seleccionando subconjuntos de estos para realizar pruebas parciales y verificar cuales de los atributos tienen más correlación con la variable de salida con el fin de reducir el conjunto atributos y con esto la complejidad del modelo. Además se

espera, en el marco del PID UTN, seguir estableciendo lazos de colaboración con industrias que provean de conjuntos de datos para evaluar el comportamiento de los modelos implementados y lograr hacer efectiva transferencia de los conocimientos adquiridos al medio.

La variedad de datos y formatos en que estos se entregan, también nos hacen concluir en la necesidad de estudiar y proponer algún tipo de normalización o desarrollo de protocolos específicos para la estructura y forma en que se comunican los datos reportados por los sensores que ayuden a agilizar el preprocesamiento para obtener más rápidos y mejores resultados, como así también poder descentralizar el desarrollo de estas herramientas.

Contribuciones

Al comienzo del desarrollo del presente proyecto, durante los trabajos enmarcados en el proyecto de investigación UTN-PID 7748 se decidió postular trabajos preliminares para las jornadas de Jóvenes Investigadores Tecnológicos - JIT 2020 - organizadas por la Universidad Tecnológica Nacional Facultad Regional Venado Tuerto, resultando éste seleccionado para ser presentado en forma virtual por quienes lo estábamos desarrollando. Se han recibido muy buenas consideraciones de parte de los revisores e interesantes aportes del auditorio que fueron tenidos en cuenta en el desarrollo de este proyecto final de carrera.

Anexo A - Instrucciones para instalar y ejecutar los prototipos

Se especifican a continuación los pasos a seguir para la instalación y ejecución del prototipo con los distintos casos de prueba expuestos en el proyecto. Si bien la mayoría de los scripts, durante el desarrollo de los diferentes trabajos, fueron escritos mediante el uso de Notebooks Jupyter en la nube (Google Colaboratory) para el caso del prototipo, este se desarrolló en entornos locales debido a la necesidad de utilizar múltiples procesos concurrentes.

Como primer requerimiento se debe instalar *python 3*[24], luego se procede a instalar las demás librerías que se utilizan en el prototipo, *matplotlib*[25], *numpy*[26], *pandas*[27] *y tensorflow*[28].

[24] *https://www.python.org/downloads/*
[25] *https://matplotlib.org/*
[26] *https://numpy.org/*
[27] *https://pandas.pydata.org/*
[28] *https://www.tensorflow.org/*

La instalación de estas librerías se puede realizar mediante el comando de consola *pip,* con algunas diferencias dependiendo de la librería, como se muestra a continuación:

- `pip install matplotlib` (Para la librería Matplotlib)
- `pip install numpy` (Para la librería Numpy)
- `pip install pandas` (Para la librería Pandas)
- `pip install --upgrade tensorflow` (Para la librería Tensorflow)

Una vez instaladas las librerías descritas hemos utilizado dos formas de ejecutar el prototipo, que se explican a continuación.

Ejecución con Pycharm

La primera opción para ejecutar el prototipo es instalar el IDE Pycharm[29], en cuyo sitio web posee toda la documentación sobre la instalación y requerimientos propios. Una vez instalado Pycharm se debe clonar el repositorio del prototipo[30] y luego de configurar el proyecto de acuerdo a la documentación de Pycharm, ejecutar el script *h_p_s_1.py* para el prototipo con el conjunto de datos TS1-4_COOLER o el script *h_p_s_2.py* para el prototipo con el conjunto de datos provisto por la industria. También se puede descargar un video[31] mostrando el prototipo y su funcionamiento en Pycharm con el conjunto de datos TS1-4_COOLER.

Ejecución con Jupyter Notebook

Otra opción es ejecutar el prototipo utilizando notebooks Jupyter[32], debiendo para esto instalar el entorno Jupyter, acción que puede realizarse mediante el comando de consola *pip*:

- `pip install notebook`

Con esta herramienta ya instalada y luego de clonar en una carpeta el repositorio del prototipo, mediante el uso de la consola de debe ingresar en la carpeta que contiene los archivos del proyecto y ejecutar el entorno Jupyter mediante el comando:

- `jupyter notebook`

[29] *https://www.jetbrains.com/es-es/pycharm/*
[30] *https://github.com/maurojp/Py_pred_sim.git*
[31] *https://www.dropbox.com/s/rh9mjuz0twr1s3x/PFC_1.mp4?dl=0*
[32] *https://jupyter.org/*

En un navegador se abre el explorador de archivos del entorno, donde con un clic podremos abrir y ejecutar celda a celda los archivos con extensión .ipynb, para el caso del prototipo con el conjunto de datos TS1-4_COOLER el archivo es *h_p_s_1.ipynb* y para el prototipo con el conjunto de datos de la industria *h_p_s_2.ipynb.*

Contenido del directorio del prototipo

Cualquiera sea el método de ejecución elegido, al clonar el prototipo, se descargan en la carpeta destino un conjunto de archivos descritos a continuación:

- *h_p_s_1.py* - Código del prototipo con el conjunto de datos TS1-4_COOLER en Python para utilizar en Pycharm.
- *h_p_s_2.py* - Código del prototipo con el conjunto de datos de la industria en Python para utilizar en Pycharm.
- *model_m.h5* - Pesos obtenidos del entrenamiento del modelo para el conjunto de datos TS1-4_COOLER.
- *model_m.json* - Modelo utilizado con el conjunto de datos TS1-4_COOLER.
- *model_MLP_Real.h5* - Pesos obtenidos del entrenamiento del modelo para el conjunto de datos de la industria.
- *model_MLP_Real.json* - Modelo utilizado con el conjunto de datos de la industria.
- *resultados_h_p_s_1.txt* - Archivo de valores separados por tabulaciones con la serie resultante de la ejecución del script *h_p_s_1.py.*
- *resultados_h_p_s_2.txt* - Archivo de valores separados por tabulaciones con la serie resultante de la ejecución del script *h_p_s_2.py.*
- *SyntheticTimeSerie_1.txt* - Series de datos sintéticas que alimentan el proceso simulador del script *h_p_s_1.py.*
- *TimeSerie_Real_Data.txt* - Series de datos reales, aportadas por la industria para alimentar el proceso simulador del script *h_p_s_2.py.*

6. Bibliografía

[1] Olarte, W. (2010). Importance of the Industrial maintence inside the processes of production. *Scientia Et Technica*, *44*, 354–356.

[2] Diez-Olivan, A., Del Ser, J., Galar, D., & Sierra, B. (2019). Data fusion and machine learning for industrial prognosis: Trends and perspectives towards Industry 4.0. *Information Fusion*, *50*, 92–111. https://doi.org/10.1016/j.inffus.2018.10.005

[3] Rich, E., & Knight, K. (1991). *Artificial Intelligence*. McGraw-Hill, Inc.

[4] Minsky, M. (1968). Pionero de la Inteligencia Artificial

[5] Copeland, B. (2020). Artificial intelligence. Encyclopedia Britannica. https://www.britannica.com/technology/artificial-intelligence

[6] Russell, S. J., Norvig, P., Manuel Corchado Rodríguez Juan, & Aguilar, J. L. (2011). *Inteligencia artificial: un enfoque moderno*. Pearson Educación.

[7] Simeone, O. (2017). A brief introduction to machine learning for engineers. *arXiv preprint arXiv:1709.02840*.

[8] Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning*. The MIT Press.

[9] Zhang, Z., & Sabuncu, M. R. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. *arXiv preprint arXiv:1805.07836*.

[10] Charniak, E. (2019). *Introduction to deep learning*. Cambridge, Massachusetts ; London, England: The MIT Press.

[11] Haykin, S. S. (2009). *Neural networks and learning machines*. Upper Saddle River, NJ: Prentice Hall.

[12] Shalev-Shwartz, S., & Ben-David, S. (2019). *Understanding machine learning: From theory to algorithms*. Cambridge: Cambridge University Press.

[13] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[14] Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

[15] Ngatchou, P., Zarei, A., & El-Sharkawi, A. (2005, November). Pareto multi objective optimization. In *Proceedings of the 13th International Conference on, Intelligent Systems Application to Power Systems* (pp. 84-91). IEEE.

[16] Team, K. *Keras documentation*. Keras. https://keras.io/.

[17] Diez-Olivan, A., Pagan, J. A., Sanz, R., & Sierra, B. (2017). Data-driven prognostics using a combination of constrained K-means clustering, fuzzy modeling and LOF-based score. *Neurocomputing*, *241*, 97–107. https://doi.org/10.1016/j.neucom.2017.02.024

[18] Fang, X., Gebraeel, N. Z., & Paynabar, K. (2017). Scalable prognostic models for large-scale condition monitoring applications. *IISE Transactions*, *49*(7), 698–710. https://doi.org/10.1080/24725854.2016.1264646

[19] Kumar, A., Chinnam, R. B., & Tseng, F. (2019). An HMM and polynomial regression based approach for remaining useful life and health state estimation of cutting tools. *Computers and Industrial Engineering*, *128*, 1008–1014. https://doi.org/10.1016/j.cie.2018.05.017

[20] J. J. A. Costello, G. M. West and S. D. J. McArthur, "Machine Learning Model for Event-Based Prognostics in Gas Circulator Condition Monitoring," in IEEE Transactions on Reliability, vol. 66, no. 4, pp. 1048-1057, Dec. 2017, doi: 10.1109/TR.2017.2727489.

[21] Guo, L., Li, N., Jia, F., Lei, Y., & Lin, J. (2017). A recurrent neural network based health indicator for remaining useful life prediction of bearings. *Neurocomputing*, *240*, 98–109. https://doi.org/10.1016/j.neucom.2017.02.045

[22] Arabnia, H., Deligiannidis, L., Yang, M. Q., American Council on Science and Education, IEEE Computer Society, & Institute of Electrical and Electronics Engineers. (n.d.). *2016 International Conference on Computational Science and Computational Intelligence : CSCI 2016 : proceedings : 15-17 December 2016, Las Vegas, Nevada, USA*.

[23] N. Helwig, E. Pignanelli and A. Schütze, "Condition monitoring of a complex hydraulic system using multivariate statistics," 2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC) Proceedings, Pisa, Italy, 2015, pp. 210-215, doi: 10.1109/I2MTC.2015.7151267.