UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

DIAGNÓSTICO DE LA DEGRADACIÓN EN REBOILERS: UN MODELO EN BASE AL APRENDIZAJE PROFUNDO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL MECÁNICO

JASSIEL ALEJANDRO RIVERA PHILLIPS

PROFESOR GUÍA:
ENRIQUE LÓPEZ DROGUETT

MIEMBROS DE LA COMISIÓN:
JUAN TAPIA FARIAS
VIVIANA MERUANE NARANJO

SANTIAGO DE CHILE
2019

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL MECÁNICO
POR: JASSIEL ALEJANDRO RIVERA PHILLIPS
FECHA: 2019
PROF. GUÍA: ENRIQUE LÓPEZ DROGUETT

DIAGNÓSTICO DE LA DEGRADACIÓN EN REBOILERS: UN MODELO EN BASE AL APRENDIZAJE PROFUNDO

En el presente trabajo de título tiene por objetivo general desarrollar un modelo en base al aprendizaje profundo que diagnostique el estado de salud de reboilers utilizados en bombas. Por otra parte, los objetivos específicos son estudiar los datos y procesarlos para que sean utilizables en el modelo, realizar un modelo de aprendizaje profundo genérico y realizar un análisis de los resultados obtenidos con los cuales se diagnosticará el estado de salud de los reboilers. La motivación para la realización de este trabajo de titulo recae en la necesidad de las empresas en disminuir sus perdidas por fallas en maquinarias las cuales tienen un monitoreo prácticamente nulo.

Dentro de los antecedentes, esta el uso de distintos tipos de redes como MLP (Multi-Layer Perceptron) que consiste en una red neuronal completamente conectada entre si. Otra de las redes a utilizar es la denominada CNN (Convolutional Neural Network), la cual se caracteriza por encontrar características en imágenes, además de de los conceptos involucrados en cada una de estas. También se realiza una comparación con un SVM (Support Vector Machine), el cual es un tipo de Supervised Machine Learning utilizado para clasificación. Por otro lado, la metodología a seguir es básicamente, recibir los datos, procesarlos, realizar el modelo, probarlo, verificar la validez del mismo y finalmente realizar el diagnostico.

Los recursos utilizados se traducen principalmente en los datos a usar en los modelos y una computadora con GPU, Linux y TensorFlow.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

Uno de los principales problemas en los sectores industriales es su escasa inversión en la detección de temprana de fallas en las maquinarias. Ya que, en general los problemas relacionados con fallas son tratados según las instrucciones del fabricante, es decir, se reemplazan piezas cada cierto tiempo, o bien, simplemente se espera a la falla de algún componente, lo cual generalmente se presenta con disminución en la producción o la detención de la máquina.

Esto muestra la importancia de desarrollar un método de identificación de fallas, que maximice el tiempo de operación de las máquinas y sea lo menos invasivo posible, para así no interferir en las labores de producción de estas mismas. Una forma de obtener distintos datos útiles sin afectar la producción de las máquinas, son los sensores, los cuales son capaces de obtener datos como presión, temperatura, vibraciones, niveles de aceite, entre otros. Estos datos por si solos no entregan ninguna información sobre si la maquina tiene una falla, por lo cual estos datos deben ser tratados de algún modo de obtener el estado de salud de la maquinaria. Uno de los métodos más efectivos de utilización de los datos de sensores en la obtención del estado de salud de una maquinaria es el aprendizaje profundo (Deep Learning).

Para comprender lo que es el aprendizaje profundo, es importante primero comprender lo que es el aprendizaje automático (Machine Learning). El aprendizaje automático es una rama de la inteligencia artificial, la cual esta involucrada en la creación de algoritmos que pueden modificarse a si mismos, sin necesidad de la intervención humana para obtener el resultado deseado, lo cual es logrado en base a alimentarse a través de datos estructurados.[21]

Por otro lado, el aprendizaje profundo (Deep Learning), utiliza capas de algoritmos, donde cada una de estas capas da una interpretación diferente de los datos ingresados. Dichas capas de algoritmos son denominadas como red neuronal, el cual posee este nombre ya que tienen como objetivo imitar la función de una red cerebral.[21]

Estos 2 métodos que utilizan la inteligencia artificial son de gran utilidad para poder determinar el estado de salud de alguna maquinaria, sin embargo, es importante conocer la forma de los datos a estudiar para ver cual es mas óptima, ya que el Machine Learning requiere datos estructurados y con etiquetas, en cambio el Deep Learning depende de las capas, las cuales pueden ser modificadas por el programador.[21]

El presente trabajo de título busca desarrollar un algoritmo en base al aprendizaje profundo que sea capaz de diagnosticar el estado de salud de reboilers en un sistema de bombeo. Para ello se utilizarán datos provenientes de sensores ubicados en distintos puntos de un reboiler de la industria papelera. Estos datos serán tratados y procesados para poder ser ingresados a un modelo de red neuronal. Es importante destacar que la base de datos recibidas servirá para realizar el modelo, pero se busca obtener un modelo genérico, es decir, que funcione con distintos conjuntos de datos.

1.1. Antecedentes básicos generales

Dentro del sistema de bombas en la industria, existe un componente denominado reboiler. Los reboilers son intercambiadores de calor de tubos, que son los encargados de evaporar el agua proveniente del generador, y de este modo controlar el nivel de vapor en el sistema.[20]

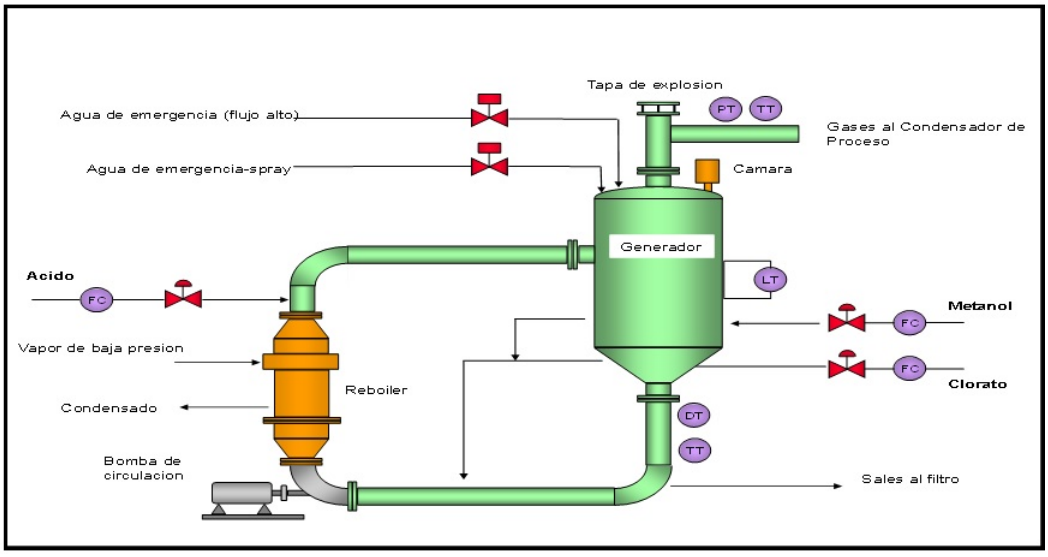

Figura 1.1: Esquema de un reboiler [19]

Uno de los problemas mas comunes de los reboilers es la formación de incrustaciones de sulfato en sus tubos, los cuales funcionan como aislante y dificultan el intercambio térmico. En casos extremos, el tubo queda completamente tapado impidiendo el paso del fluido intercambiador.[20]

Figura 1.2: Reboiler con tubos tapados [19]

Los datos a utilizar para la realización del modelo se obtendrán de sensores colocados en distintos puntos de los reboilers a estudiar, los cuales pertenecen a una empresa del rubro papelero. Los datos obtenidos de los sensores vendrán sin un procesamiento previo, por lo cual, para utilizarlos en el modelo, estos deberán ser tratados y procesados para poder ser ingresados al modelo.

1.2. Motivación

Las bombas industriales son un pilar fundamental en el funcionamiento de distintas industrias, por lo cual, es de vital importancia conocer el estado de salud de estas, y de este modo planificar el mantenimiento de estas y evitar bajas en la producción y perdidas monetarias asociadas a fallas en las bombas.

De esta forma, la motivación consiste en realizar un modelo genérico (que sirva para distintos conjuntos de datos, y no solo un reboiler en específico) que estudie el estado de salud de los reboilers, para así, ayudar a la industria a disminuir sus perdidas producidas por fallas inesperadas.

1.3. Objetivos

1.3.1. Objetivo general

- Desarrollar algoritmo capaz de diagnosticar el estado de salud de reboilers de bombas industriales.

1.3.2. Objetivos específicos

- Estudiar datos sin procesar y observar su comportamiento en busca de alguna tendencia o utilidad de estos.
- Procesar los datos, de forma de que puedan ser utilizables en un modelo de aprendizaje profundo.
- Realizar un modelo de aprendizaje profundo, por medio de la utilización de redes MLP y CNN.
- Desarrollar dentro un modelo, una forma de verificar la validez del modelo y así observar si es capaz de recibir distintos conjuntos de datos (evitar sobre-ajuste).
- Comparar los resultados de las redes neuronales con un SVM.
- Realizar un análisis de los resultados obtenidos, y con ellos lograr identificar cual es el mejor método para realizar el diagnóstico del estado del reboiler.

1.4. Alcances

El presente trabajo de titulo tiene como alcance el diagnostico del estado de salud actual de los reboilers en un sistema de bombas industriales, por lo cual, no se realizara prognosis, es decir, predecir el estado de salud de los reboilers en un futuro.

Capítulo 2

Antecendetes

El modelo a realizar para la memoria tiene distintos tipos tipos de redes neuronales involucradas, las cuales serán la clásica red artificial neuronal (ANN o MLP) y las redes convolucionales (CNN). Además se utilizará una SVM para comparar los resultados con los modelos anteriormente nombrados. También es importante conocer varios conceptos relacionados con el tema, para así al momento de explicar resultados o analizar el modelo, estos sean comprendidos por el lector. En el presente capitulo se abordaran dichos temas.

2.1. Conceptos generales

2.1.1. Overfitting (sobre ajuste)

El overfitting o sobre ajuste consiste en que el modelo solo reconoce los datos particulares ingresados, y en el momento que se le ingresen datos nuevos, el modelo sera incapaz de identificar el comportamiento de estos. Este problema tiene diversas causas, donde las principales son cuando en modelo es muy complejo, el tamaño de los datos de entrenamiento es muy pequeño y/o la existencia de muchos parámetros de aprendizaje [1][3].

2.1.2. Underfitting (sub ajuste)

A diferencia del caso anterior, el underfitting o sub ajuste consiste en que el modelo es incapaz de reconocer la estructura de los datos de entrenamiento, por lo cual, no puede realizar un diagnostico correcto. Este fallo generalmente es producido por un modelo demasiado simple. En la siguiente imagen se muestra de forma ilustrativa los 2 conceptos anteriormente nombrados [1][3].

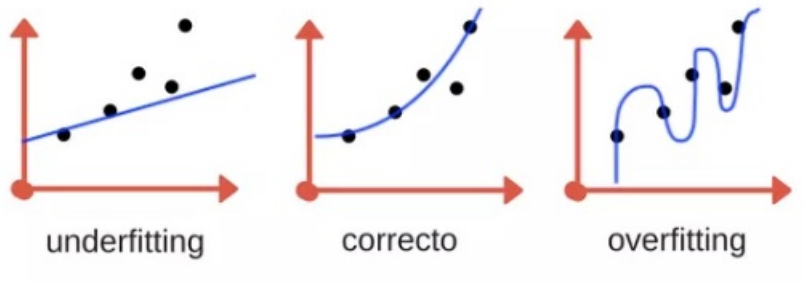

Figura 2.1: Comparación estados de los modelos [3]

2.1.3. Validación cruzada

Cuando los datos de entrenamiento son escasos, de tal modo que es inviable realizar construir un set de validación adecuado. Una solución es la utilización de la validación cruzada (Cross-Validation) es una técnica que divide los datos de entrenamiento en K sub-conjuntos no superpuestos. Luego el set de entrenamiento y validación se ejecutan K veces, donde cada vez que se entrena se hace en K-1 subconjuntos y se valida un subconjunto diferente (el no usado en el entrenamiento). Finalmente el error de los sets de entrenamiento y validación se calcula promediando los resultados de los K experimentos [19].

2.1.4. Matriz de confusión

Una matriz de confusión se define como una matriz de k x k, donde cada columna corresponde a la etiqueta predicha y cada fila a la etiqueta real del modelo. Esta matriz nos indica el rendimiento del modelo. Las matrices de confusión más comunes son de 2x2 como se observa en la siguiente imagen [1][6][18].

		PREDICTED CLASS	
		Class 1	Class 2
ACTUAL CLASS	Class 1	True Positive (TP)	False Negative (FN)
	Class 2	False Positive (FP)	True Negative (TN)

Figura 2.2: Composición matriz de confusión 2x2 [1]

Donde:

- TP (True Positive): Son las muestras son correctamente clasificadas como positivas.
- FN (False Negative): Son las muestras incorrectamente clasificadas como negativas.
- FP (False Positive): Son las muestras incorrectamente clasificadas como positivas.
- TN (True Negative): Son las muestras correctamente clasificadas como negativas.

2.1.5. Accuracy

El accuracy es una métrica utilizada para evaluar el desempeño de un modelo. Esta consiste en el cociente entre las predicciones que el modelo realizo correctamente y el numero total de predicciones, por lo cual, el accuracy no indica el porcentaje de datos correctamente clasificados. Dentro del desarrollo de un modelo es de vital importancia obtener valores cercanos a 1, es decir, que clasifique con la mínima cantidad de errores posibles [1]. Utilizando los valores obtenidos de la matriz de confusión, el accuracy se puede calcular de la siguiente forma:

$$Accuracy = \frac{TP + TN}{TP + FN + FP + TN} \times 100 \tag{2.1}$$

2.1.6. Batch size y epoch

El batch size es la cantidad de muestras que se le entregara a la red para que esta entrene [1]. Por ejemplo si tenemos 1000 datos y fijamos un batch size de 100, significa que los 100 primeros datos pasaran por la red para ser entrenados, inmediatamente después desde el 101 al 200 entraran en la red para ser entrenados y así hasta llegar al 1000, de este modo cuando pasan todos los datos por la red, se cumple 1 época (epoch). Las épocas se definen como el numero de veces que a la red es entrenada.

2.2. Redes neuronales

Los seres humanos a lo largo de sus vidas aprenden una gran cantidad de tópicos, empezando desde el lenguaje hasta conocimientos tan complejos como para construir un robot. Los diversos aprendizajes obtenidos a lo largo de la vida son debido a las redes neuronales ubicadas en nuestro cerebro. De igual forma que nuestro cerebro, las redes neuronales pueden ser replicadas en modelos computacionales, y con ellas, en base a un conjunto de datos, aprender patrones y poder diagnosticar el estado de salud de distintas maquinarias.

Las redes neuronales artificiales se definen como un sistema de procesamiento de datos por medio de elementos (neuronas) conectados entre si, tal como un cerebro humano.

Las redes neuronales tienen como ventaja que caracterizan modelos no lineales, modelan fenómenos complejos sin tener una base de datos anterior, no hay que suponer distribuciones de probabilidad y es adecuado para problemas multidimensionales.

2.2.1. MLP (Multi-Layer Perceptron)

Las redes MLP son aquellas que están constituidas por capas, donde cada capa esta compuesta por nodos, donde cada nodo esta completamente conectado con los nodos de la capa posterior. Este tipo de red esta compuesta de 3 partes, una capa de entrada, una o más capas ocultas y una capa de salida. A continuación se explica la estructura, funcionamiento y algunos conceptos básicos de este tipo de red [1].

2.2.1.1. Estructura y conceptos básicos

Para conocer una red neuronal es importante conocer su estructura y que la compone, a continuación se observan la estructura principal de red y algunos conceptos básicos:

- Capa entrada: Es la capa que recibe la información a modelar [1][7].
- Pesos: Asociada a cada conexión de neurona artificial, es la intensidad de señal, similar a la señal sináptica del cerebro humano [1][7].
- Sesgo: Asociada a cada neurona, es la encargada de ajustar la señal de la neurona [1][7].
- Función de activación: Es aquella que le quita la linealidad a los datos que ingresan a la neurona [1][7].
- Capa salida: Es la capa que contiene las neuronas de salida del modelo, las cuales realizan la clasificación de este. Cada 1 de las neuronas tiene una característica, por ejemplo, un rodamiento con 2 modos de falla (A, B), las neuronas de salida serian 3, la primera con A, la segunda con B y la tercera con estado sano [1][7].

- Capa oculta: Son las capas ubicadas entre la capa de entrada y la capa de salida. En ellas se realizan la mayoría de los cálculos del modelo. Como su nombre lo indica, estas capas son 'ocultas al ojo humano', es decir, no se observan los resultados parciales obtenidos en estas capas [1][7].

En la siguiente es posible observar algunos de los conceptos nombrados anteriormente:

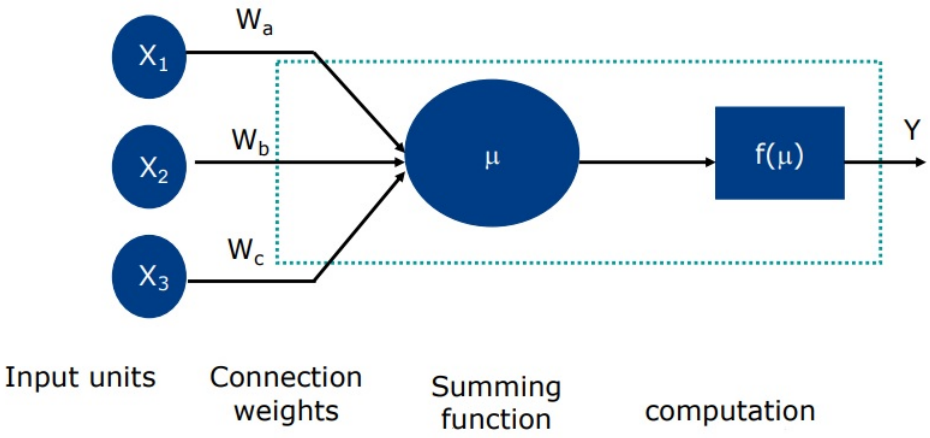

Figura 2.3: Elementos de una red neuronal [1]

2.2.1.2. Funcionamiento

El funcionamiento de la red neuronal se puede definir en los siguientes pasos:

1. **Inicializar los pesos:** Esto sucede en la primera época de funcionamiento del modelo.

2. **Calcular los valores de entrada a la primera capa oculta:** Antes de entrar a las capas ocultas, los pesos y el sesgo se relacionan por medio de una función lineal del tipo Ax+B, con A el peso, x la neurona (que tendrá el valor de los datos ingresados) y b el sesgo. Todas las neuronas de la capa de entrada se conectan con cada una de las neuronas de la capa oculta, por lo cual el valor que ingresa a dichas neuronas de la capa oculta es la sumatoria de las distintas funciones lineales obtenidas [1].

3. **Cálculo de valores de salida de las capas ocultas:** Al valor ingresado anteriormente, en cada neurona se le aplica una función de activación, y de esa forma cada neurona obtiene un valor. Estas neuronas se conectan con otra capa oculta, donde al igual que en el punto anterior, todas las neuronas de esta capa se conectan con cada una de las neuronas de la capa sub-siguiente por medio de una función lineal entre su valor y el peso de la conexión con la nueva neurona. Después de pasar por todas las capas ocultas, la ultima capa oculta se conecta con la capa de salida de la misma manera [1][2].

4. **Cálculo de valores de la capa de salida:** La capa de salida recibe las señales de la ultima capa oculta, y por medio de una función de activación (distinta a las de las capas ocultas), realiza la predicción del modelo [1].

5. **Cálculo de error y actualización de los pesos:** Luego de obtener los valores de la capa de salida, se calculan los errores y se actualizan los pesos de las neuronas, esto por medio del algoritmo Backpropagation [1].

6. **Iteración:** Este proceso se repite el numero de épocas asignado al modelo.

7. **Revisión del modelo:** Se revisa que finalizadas todas las épocas, el modelo tenga un accuracy y un error mínimo, y que además no este overffited.

La siguiente imagen muestra el funcionamiento de 1 neurona, con valores numéricos en forma de ejemplo:

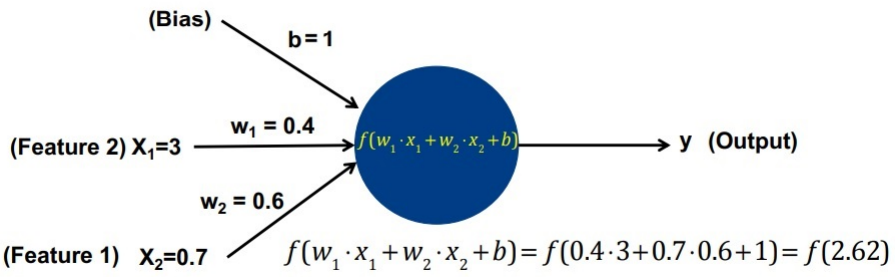

Figura 2.4: Ejemplo funcionamiento de una neurona [1]

Como se dijo anteriormente, dentro de la neurona se le aplica una función de activación a la función linea, dicha función de activación va a variar dependiendo las condiciones de los datos a trabajar, entre las funciones de activación más usadas se encuentran:

- Identidad: No es de mucha utilidad en modelos complejos, pues con esta se mantiene la linealidad de la función. Esto no es conveniente ya que el aprendizaje de la red usa el gradiente de la función, y la derivada de una función lineal es una constante [1][8].

- Step: Esta función se caracteriza por ser binaria, donde vale 1 si el parámetro de la función es mayor a cero, y 0 en todos los otros casos. Esta función tiene como inconveniente que solo da como resultados 0 y 1, esto provoca que los pesos de las neuronas sean irrelevantes. Otro problema es el mismo de la identidad, el cálculo del gradiente para el backpropagation [1][8].

- Sigmoid: Es una función que toma valores entre 0 y 1. El problema que tiene este tipo de función es que en los valores cercanos a 0 y 1 tiene variaciones muy pequeñas, esto provoca que los gradientes serán pequeños y provocara un mal aprendizaje. De igual modo esta función es utilizable en algunos tipos específicos de modelos [1][8].

- Tanh: Esta función toma valor entre -1 y 1. Se comporta de manera muy similar a la función sigmoid en su comportamiento, aún así, su gradiente en los extremos es más fuerte que la sigmoid, por lo que tiene errores un poco menores en el gradiente causados por eso. Es utilizada cuando el dataset esta normalizado entre -1 y 1 [1][8].

- ReLU: Esta función toma valores entre 0 y infinito, esto evita los problemas de gradiente anteriormente nombrado, ya que aunque gráficamente se vea lineal en la zona positiva, no lo es, y al no tener cota superior, el gradiente no sera más pequeño en zonas especificas. Esta es muy utilizada, especialmente cuando los datos están normalizados entre 0 y 1 [1][8].

- SoftMax: Esta función 'aplana' la salida para que los valores estén entre 0 y 1, aún más, hace que la suma de las salidas sea igual a 1. Este tipo de función es óptima para las neuronas de salida de la red, ya que al realizar diagnostico estamos calculando una probabilidad de que este en dicho estado, por lo cual la suma de las salidas debe ser 1 [1][8].

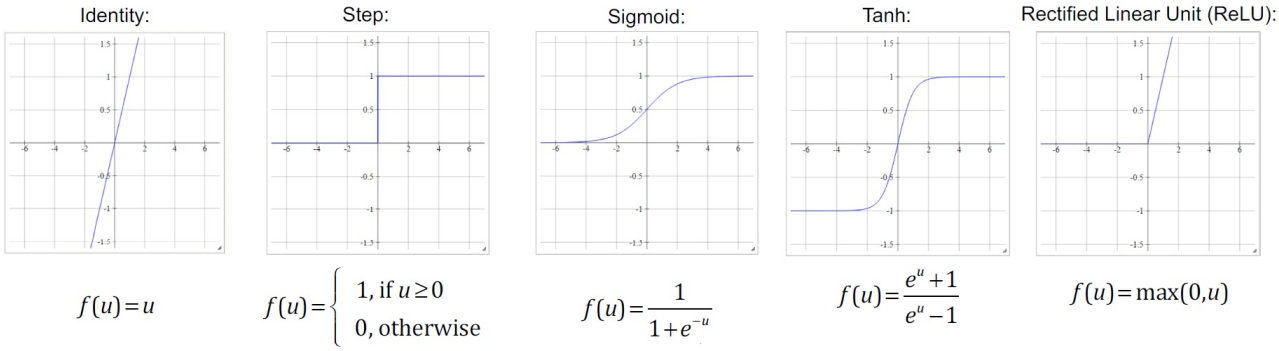

Figura 2.5: Resumen funciones activación [1]

2.2.1.3. Backpropagation

El Backpropagation es un algoritmo que propaga hacia atrás los errores en la red, esto por medio de la técnica del gradiente descendiente, la cual, consiste en evaluar la pendiente en un punto esto por medio del cálculo del gradiente negativo, ya que el gradiente positivo muestra el incremento de la pendiente. Luego de llegar al nuevo punto, este proceso se repite de forma iterativa hasta obtener un gradiente negativo que sea tan ínfimo que no suponga mayor cambio a la función, es decir, el mínimo local. Este método se utiliza principalmente en funciones de costo no convexas, es decir, que poseen distintos puntos donde la pendiente vale cero y no solo el mínimo global [1][18].

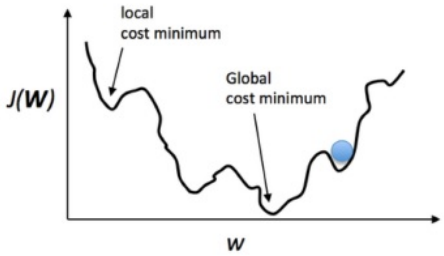

Figura 2.6: Gráfico función no convexa [1]

El Backpropagation tiene por objetivo actualizar los pesos de las neuronas y de este modo minimizar el error y mejorar el desempeño de la red neuronal [1]. Este tipo de entrenamiento que tienen las redes neuronales tiene los siguientes pasos a seguir:

1. Inicializar los pesos: En la primera época, se inicializan los pesos de las neuronas de forma aleatoria [1].

2. Obtención valores de salida: Posterior a recorrer toda la red, se obtiene el valor de la ultima capa de la red [1].

3. cálculo del error y gradiente: Esto se hace comparando el valor obtenido de cada neurona de salida con el valor real deseado, para esto usamos la formula del error cuadrático medio [1].

$$E_z = \sum_{\mathrm{i}=1}^{N} (t_{\mathrm{i}} - y_{\mathrm{i}}) \tag{2.2}$$

   Con:

   - E_z error en la neurona z.
   - t_{i} el valor real.
   - y_{i} el valor obtenido del modelo.

   Ahora bien, utilizando la técnica del gradiente descendiente, nombrada anteriormente, el error puede reducirse ajustando cada peso en la dirección:

$$A = -\sum_{z=1}^{Z} \frac{\mathrm{d}E_z}{\mathrm{d}w_{j\mathrm{i}}} \tag{2.3}$$

4. Actualización de los pesos: Para actualizar los pesos, matemáticamente consiste en aplicar la regla de la cadena al ajuste A (ecuación 2.3) y agregarle una taza de aprendizaje η [1]. Esto de traduce de la siguiente manera:

$$w_{n\mathrm{i}}^{'} = w_{n\mathrm{i}} + \eta\delta_n \frac{\mathrm{d}f_n(\mathrm{e})}{\mathrm{de}} x_n \tag{2.4}$$

Donde:

- $w_{in}^{'}$ es el peso actualizado.
- w_{in} es el peso anterior.
- η es el learning rate (taza aprendizaje).
- δ_n es el error de la neurona n.
- $\frac{\mathrm{d}f_n(\mathrm{e})}{\mathrm{de}}$ es la derivada de la función de activación.

.

5. Iteración del proceso: Luego este proceso se repite de forma iterativa, para de que de esta forma la red vaya mejorando y así obtener un error mínimo.

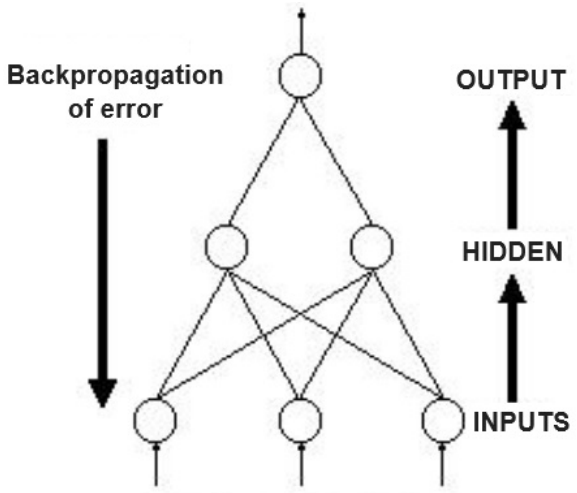

Figura 2.7: Backpropagation [1]

2.2.2. CNN (Convolutional Neural Network)

Es una red neuronal de múltiples capas, las cuales se utilizan para encontrar patrones y características, principalmente en imágenes, ya que a diferencia de las MLP, a este tipo de red neuronal se le ingresa información en forma de matriz (imagen) [11].

Para comprender el funcionamiento de una CNN es necesario entender como funciona una convolución. El termino convolución se refiere a la combinación matemática de dos funciones para producir una tercera función. En el caso de las CNN,la convolución se realiza entre los datos de entrada y el filtro, para de este modo obtener un mapa de características [21].

A modo de ejemplo del funcionamiento de una convolución en una red CNN podemos considerar lo siguiente. Suponer una imagen de entrada de 5x5 y utilizar un filtro de tamaño 3x3, este filtro se recorrerá toda imagen, y en cada posición que este el filtro de 3x3 se multiplica termino a termino y se suman los valores obtenidos. En la siguiente imagen se muestra el ejemplo anteriormente nombrado con números:

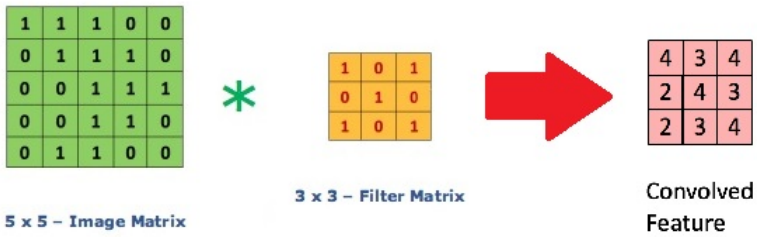

Figura 2.8: Ejemplo convolución [9]

Así, cuando el filtro esta en la primera posición se realiza una operación de (1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 0x1 + 0x0 + 1x1) = 4, luego el filtro se mueve a la derecha y vuelve a realizar la misma operación lineal entre la imagen y el filtro, de ese modo hasta que el filtro recorra toda la imagen.

Las distintas capas de la CNN funcionan del mismo modo, y mientras más profundo sea el modelo mejores características se obtendrán de la imagen. aún así, un modelo CNN no puede clasificar, por lo cual a la salida de la ultima capa de la CNN se debe agregar una red MLP. La información proveniente de una CNN esta en forma de imagen, pero la MLP solo recibe datos en forma de vector, por lo cual las imágenes se 'aplanan' para formar un vector y ser ingresados a la MLP.

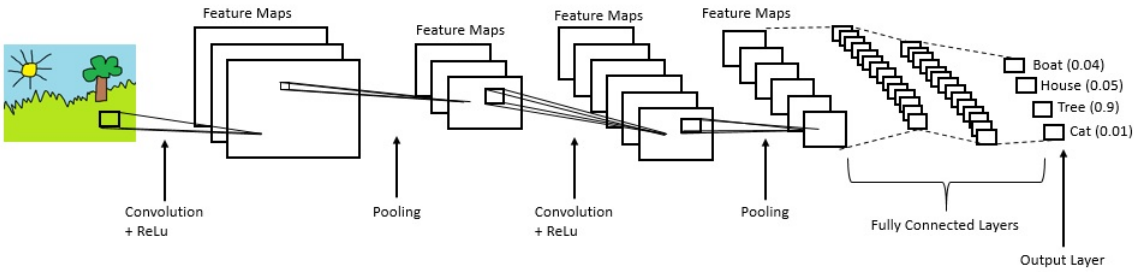

Figura 2.9: Esquema funcionamiento red CNN [9]

De la figura 2.9 es posible observar que del resultado de la convolución se obtiene una disminución de la dimensionalidad de la matriz (imagen), esto se puede corregir utilizando padding y que así la imagen de salida de la capa tenga el mismo tamaño que la de entrada. El avance que realiza el filtro (stride) es ajustable, aunque usualmente se utiliza el valor por defecto (avanzar 1 espacio). Cuando la imagen es muy grande, el procesamiento es lento, por lo cual se le pueden extraer sus características principales a través del Max Pooling.

2.2.2.1. Kernel size (tamaño filtro)

Un filtro es una matriz de pesos con el cual se convoluciona con la imagen de entrada. Este filtro da una medida de que tan cercana es una característica al parche de entrada. Los filtros mas pequeños recopilan información de forma local, mientras que los mas grandes recopilan información global y representativa [21].

2.2.2.2. Padding

El padding es una técnica que agrega ceros a las filas y columnas de la imagen, esto con el objetivo de mantener los tamaños espaciales constantes después de la convolución y de este modo mejorar el rendimiento de las siguientes convoluciones, ya que con esta técnica se retiene la información de los bordes [21].

2.2.2.3. Stride

Es el tamaño del paso del filtro para recorrer la imagen de entrada despues de cada convolución. Por defecto su valor es de 1 [21] .

2.2.2.4. Max Pooling

Es una técnica de reducción de dimensionalidad, esta consiste en identificar la propiedad de mayor valor en un grupo y solo mantener dicho valor. En la siguiente imagen se observa de forma más clara [9]:

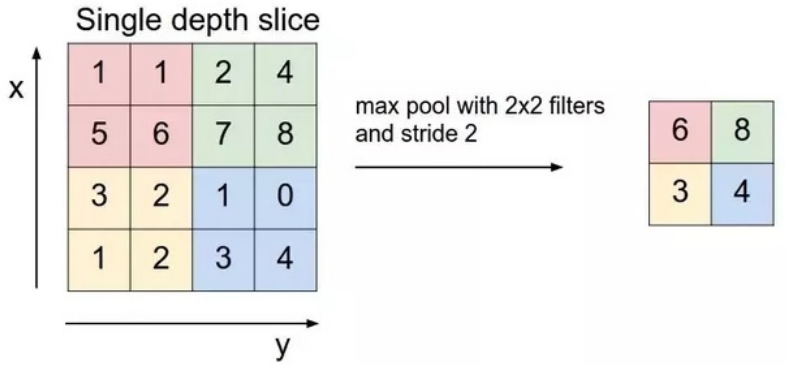

Figura 2.10: Max Pooling [9]

2.3. SVM (Support Vector Machine)

El Super Vector Machine (SVM), es un tipo de Supervised Machine Learning basado en la teoría de aprendizaje estadístico, que puede ser utilizado en clasificación lineal o no lineal, regresiones y en detección de valores atípicos [25][26].

La clasificación de los datos se realiza por medio de un hiper-plano, el cual separa en 2 clases distintas el conjunto de datos a estudiar.

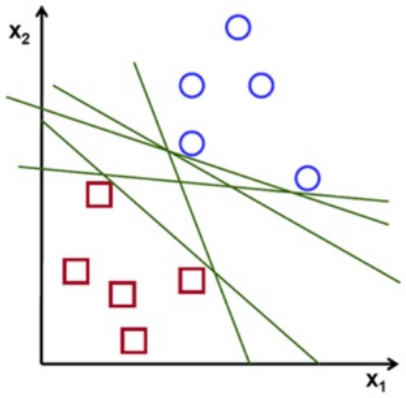

Figura 2.11: Hiper-planos capaces de separar las clases [24]

Como se observa en la figura 2.11, existen muchos hiper-planos capaces de separar en 2 clases distintas el conjunto de datos. Aun así, existe un hiper-plano óptimo, el cual maximiza el margen entre la decisión limite que separa las 2 clases, obtener dicho hiper-plano es el objetivo del SVM [24][25].

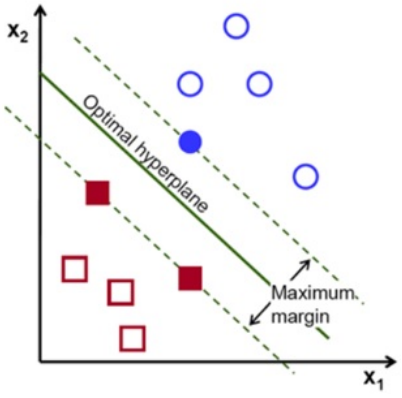

Figura 2.12: Hiper-plano óptimo [24]

Para encontrar este hiper-plano, se tiene un conjunto de vectores x $\in R^n$, asociables a cualquiera de las 2 clases y $\in$ +1, -1, se desea encontrar una función f(x) que permita clasificar los datos en una de las clases y se busca un hiper-plano que tenga la forma de la siguiente ecuación [26]:

$$w^T x + b = 0 \tag{2.5}$$

En la figura 2.12 se observa un ejemplo donde cada vector x = (x1,x2) se asocia a cada una de las dos clases (+1, -1). En la misma figura, es posible observar 2 rectas con una linea punteada que acotan el margen del hiper-plano, estas rectas se denominan Vectores Soportes (Support Vectors). Posterior a entrenada la maquina, se obtiene una función de decisión, con la cual en base a los vectores (x1, x2) entrega un valor escalar [26]:

$$\mathrm{d}(x, w, b) = w^T x + b = \sum_{\mathrm{i}=1}^{n} w_{\mathrm{i}} x_{\mathrm{i}} + b \tag{2.6}$$

Como los valores de w son infinitos, se busca aquel que maximice el margen entre ambas clases, el cual es $\frac{2}{\|w\|}$, que es la distancia entre que separa a los vectores soportes de clases distintas y se obtiene de la siguiente ecuación [26]:

$$min \frac{1}{2} w^T w \tag{2.7}$$

Esto se traduce en que para cada Vector Soporte se cumple que $y_{\mathrm{i}}[w^T x_{\mathrm{i}} + b] = 1$. De esta forma podemos definir la siguiente restricción para la ecuación 2.7 [26]:

$$y_{\mathrm{i}}[w^T x_{\mathrm{i}} + b] \geq 1 \tag{2.8}$$

Es posible escribir el Lagrangeano del problema, introduciendo un multiplicador α_{i} asociado a cada vector x_{i}, obteniendo así la siguiente ecuación [26]:

$$L(w, b, \alpha) = \frac{1}{2} w^T w = \sum_{\mathrm{i}=1}^{l} \alpha_{\mathrm{i}} y_{\mathrm{i}} [w^T x_{\mathrm{i}} + b] - 1 \tag{2.9}$$

Para resolver el problema se utilizan las condiciones Karush-Khun-Tucker (KKT), las cuales son suficientes y necesarias para la maximización de la ecuación 2.9. Así maximizando α_{i} y minimizando w y b se obtiene [26]:

$$\frac{\partial L}{\partial w_0} = 0 \Rightarrow w_0 = \sum_{\mathrm{i}=1}^{l} \alpha_{\mathrm{i}} y_{\mathrm{i}} x_{\mathrm{i}} \tag{2.10}$$

$$\frac{\partial L}{\partial b_0} = 0 \Rightarrow \sum_{\mathrm{i}=1}^{l} \alpha_{\mathrm{i}} y_{\mathrm{i}} = 0 \tag{2.11}$$

$$\alpha_{\mathrm{i}} y_{\mathrm{i}} [w^T x_{\mathrm{i}} + b] - 1] = 0, \mathrm{i} = 1, ..l \tag{2.12}$$

Reemplazando en el Lagrangeano primal, para obtener el dual, y a su vez minimizando el dual, se obtienen los siguientes resultados [26]:

$$w_0 = \sum_{\mathrm{i}=1}^{l} \alpha_{\mathrm{i}} y_{\mathrm{i}} x_{\mathrm{i}} \tag{2.13}$$

$$b_0 = \frac{1}{N_{SV}} \sum_{\mathrm{i}=1}^{N_{SV}} (y_s - x_s^T w_0), s = 1, ...N_{SV} \tag{2.14}$$

Donde N_{SV} representa el numero de SV (Suppot Vectors) encontrados en la optimización. De esta forma se obtiene que el hiper-plano de decisión D(x) [26]:

$$D(x) = \sum_{\mathrm{i}=1}^{l} w_0 x_{\mathrm{i}} + b_0 = \sum_{\mathrm{i}=1}^{l} y_{\mathrm{i}} \alpha_{\mathrm{i}} x^T x + b_0 \tag{2.15}$$

Capítulo 3

Metodología

La metodología a utilizar se sintetiza en los siguientes pasos:

1. **Recibir los datos**: Datos provenientes de sensores ubicados en distintas partes del reboiler a estudiar ubicada en una empresa del rubro minero.

2. **Procesar los datos**: Los datos no necesariamente vendrán en las condiciones óptimas de trabajo, por ello se le realizan distintos procesos para que puedan ser utilizables en el modelo (obtener características, etiquetarlos, dividir set de entrenamiento, testeo, etc.).

3. **Realizar modelo**: Se realizara un modelo de redes neuronales que logre diagnosticar el estado de salud del reboiler.

4. **Hacer correr el modelo**: Al modelo se le ingresan los datos ya procesados con los cuales se corre el modelo. Se observa si los resultados obtenidos cumplen los requisitos mínimos de diagnostico (Accuracy), y se realizan las modificaciones necesarias hasta obtener un modelo que diagnostique de forma aceptable.

5. **Validar el modelo**: Revisar que el modelo sea genérico, es decir, que funcione con distintos conjuntos de datos y no solo con los datos suministrados por la empresa, por lo cual se tendrán datos (de los mismos sensores) pero que nunca hayan sido observados por el modelo, para así validar el modelo.

6. **Realizar diagnostico**: Ya al haber comprobado la generalidad del modelo, se procede a realizar el diagnostico del estado actual de salud del reboiler en base a los datos entregados por una empresa del rubro minero.

El siguiente diagrama muestra un resumen de la metodología a seguir:

Figura 3.1: Metodología a seguir

3.1. Recursos

Para la presente memoria no se requerirán recursos monetarios, por lo cual, no existen los recursos pecuniarios.

3.1.1. No pecuniarios

- Datos provenientes de los sensores ubicados en el reboiler a estudiar
- Computadoras con GPU, Linux y TensorFlow ubicadas en Beauchef 851.

Capítulo 4

Desarrollo

4.1. Datos sin procesar

Los datos obtenidos del reboiler para la implementación del modelo tienen las siguientes características:

- Se utilizaron 118 sensores colocados a lo largo del del reboiler.
- Datos capturados desde el año 2.006 al año 2.018, lo que se traduce en 52.013.
- Intervalo de toma de datos de 2 horas.
- Datos indexados de la forma fecha-hora.
- Existencia de sensores apagados o no utilizados en la muestra.
- Existencia de varios periodos de tiempo sin datos.
- Datos aislados fuera de parámetros (valores del tipo 10^{33}).
- La mayoría de los datos no están etiquetados, es decir, no se conoce su estado de salud (solo hay 2.299 datos con etiqueta).

A estos datos no le es posible aplicar un modelo, ya que además de tener muchos vacíos, no tiene etiqueta, por lo cual deben ser procesados antes de utilizarse.

4.2. Procesamiento de datos

Como se menciona anteriormente, los datos no vienen en condiciones aptas para el modelo, por lo cual se debe realizar una limpieza de estos. La limpieza consiste en algunos puntos, los cuales se mencionan a continuación:

- Se elimina la columna que contiene fecha-hora.
- Se eliminan las columnas que tengan los sensores apagados, es decir, que no tengan registros de datos.
- Se considera que un sensor tiene datos suficientes para ingresar al modelo cuando supera el 75 % del total, en caso contrario el sensor es eliminado.
- Posterior a los pasos anteriores, se eliminan las filas que no tengan registro de datos (NaN) y valores fuera de rango.

De esta forma se obtiene una matriz sin datos Nan o saltos temporales y con todos los sensores funcionales, ahora es importante realizar un método para poder etiquetar los datos, para así finalmente tener todos los componentes necesarios para ingresar los datos al modelo.

4.3. Etiquetado de los datos

Para etiquetar los datos se procede a utilizar 2 técnicas. La primera de ellas es el PCA (Principal Component Analysis).

El PCA es una técnica de reducción de dimensionalidad de un conjunto de datos que posee un gran numero de variables interrelacionadas. Al momento de reducir la dimensionalidad, se busca conservar la mayor cantidad posible de varianza del conjunto de datos, esto se logra a transformando los datos en un nuevo set de variables, denominado 'Principal Components' (PCs), que no están correlacionados entre si (independientes) y están ordenados de forma que los primeros retengan la mayor cantidad de variación presente en todas las variables originales [22].

Posterior a utilizar el PCA, se utiliza el método del DB-SCAN (Density-Based Spatial Clustering of Applications with Noise).

El DB-SCAN es un algoritmo basado en la agrupación de datos basado en la densidad, donde objetos similares se agrupan en clusters dependiendo de su densidad dentro de un conjunto de datos dado. Los clusters se definen como regiones en las cuales los datos son mas densos, separados por regiones de menor concentración [23]. De esta forma, utilizando el DB-SCAN es posible agrupar en clusters datos con semejantes características.

Estas 2 técnicas son de vital importancia para etiquetar los datos, ya que con la primera logramos reducir la dimensionalidad del problema y con la segunda podemos etiquetar los datos (reconocer dentro del cluster y fuera del cluster). A la hora de realizar los métodos anteriormente nombrados, se procedió de la siguiente manera:

- Inicialmente se comienza utilizando el PCA con un número de variables equivalente al número de sensores y con ello obtener un gráfico que nos muestre que cantidad de sensores son necesarios para obtener una mínima varianza acumulada de 80 %. De ese modo se obtiene el siguiente gráfico:

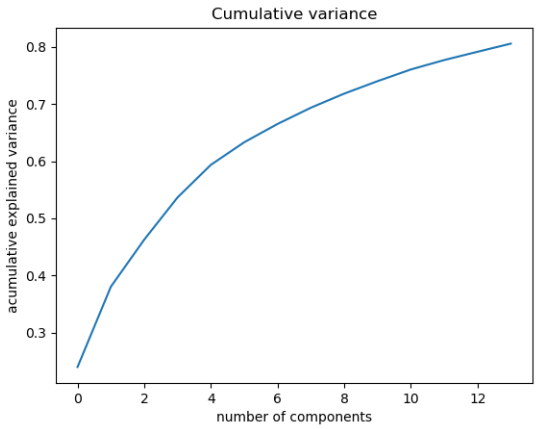

Figura 4.1: Varianza acumulada en PCA

  Del gráfico podemos observar que la mínima cantidad de componentes (sensores) que debemos utilizar para tener dicha varianza acumulada es de 14.

- Luego de obtener la cantidad mínima de sensores para obtener una muestra representativa, se procede a correr nuevamente el PCA, esta vez con el número de componentes obtenidos (14) para de ese modo reducir la dimensionalidad.

- Posterior a reducida la dimensionalidad del problema, se procede a etiquetar, lo cual se realiza con la ayuda del DB-SCAN. Esta técnica consiste en agrupar datos con semejantes características en clusters. A la hora de realizar el DB-SCAN se debe definir la distancia máxima entre los 2 puntos que se considera para que 2 puntos sean del mismo vecindario (eps) y la cantidad de muestras (peso total) en un vecindario para que un punto se considere punto central del vecindario (min samples).

- Las etiquetas existentes de la empresa (2.299), basadas en sus rangos de operación, se comparan con la traza correspondiente en el tiempo de las etiquetas obtenidas a través del DB-SCAN, y se hace una matriz de confusión para observar la precisión de este.

- Se iteran los 2 parámetros anteriormente nombrados (eps y min samples), hasta obtener una matriz de confusión aceptable, es decir, minimizando los falsos positivos y buscando una precisión mínima del 70 %. De ese modo utilizando eps= 4,1 y min samples= 7.000, se obtiene la siguiente matriz de confusión:

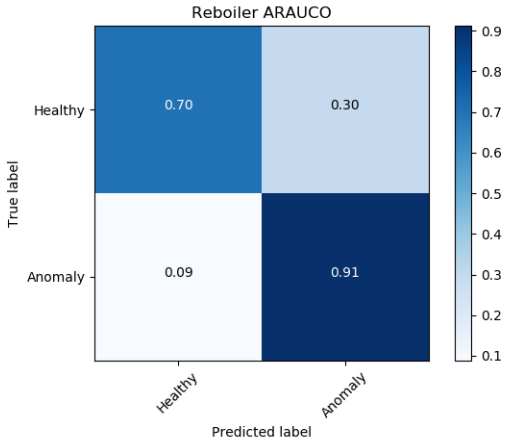

Figura 4.2: Matriz de confusión normalizada entre etiquetas empresa vs predecidas

- Se procede a juntar los datos con sus respectivas etiquetas.

- A la matriz de datos ya etiquetados le se proceden a separar en X_{test}, X_{train}, Y_{test} y Y_{train} de forma aleatoria (para no obtener siempre el mismo resultado).

- Finalmente los distintos sets son guardados en un archivo nuevo, para su posterior utilización en el modelo.

4.4. Experimentos

Luego de realizar los procesos anteriores, se obtiene una matriz apta para su utilización, sin saltos temporales, sensores apagados y los datos etiquetados, por lo cual, ahora es posible definir los experimentos a realizar.

4.4.1. Experimento 1: Red MLP

El primero de los experimentos a realizar consiste en diseñar una red MLP. A esta red se le ingresaran los datos y se observaran varias medidas de precisión como el accuracy, matriz de confusión y gráficos de perdida. Esta red tendrá ciertas características, las cuales son las siguientes:

- Número de capas: Este hiper-parámetro fue seleccionado por medio de realizar "ensayo y error", ya que se probaron distintos números de capas y se concluyo que el numero óptimo de capas para este modelo era 10.

- Número de neuronas por capas: A lo largo de las capas el número de neuronas va descendiendo, para que de este modo la red aprenda a lo largo de las capas [14][15][16]. El número inicial de neuronas es de 20000 y esta va descendiendo hasta 7 neuronas en la ultima capa.

- Número de épocas: Se testeara la red con 100 y 200 épocas, para así observar la influencia de este parámetro en la red.

- Función de activación: Se utilizara ReLU, ya que los datos están normalizados entre 0 y 1. En la neurona final se utiliza la función sigmoid, ya que esta nos sirve como clasificador binario.

- Dropout: Se utiliza un valor de 20 %, es decir, ignoramos el 20 % de las neuronas.

- Batch Size: Se utilizan distintos batch sizes para así observar la influencia de este parámetro en la red. De igual modo, los batch size utilizados siempre sera divisor del numero total de datos a entrenar.

4.4.2. Experimento 2: Red CNN

El segundo de los experimentos a realizar consiste en diseñar una red CNN. A esta red, al igual que a la anterior, se le ingresaran los datos y se observaran varias medidas de precisión como el accuracy, matriz de confusión y gráficos de perdida. Una gran diferencia de este tipo red con la anterior es la forma en que se le ingresan los datos, mientras que en la anterior se le ingresan en forma de vector, en este caso se le pueden ingresar imágenes en 1D, 2D y hasta 3D [18]. En el caso del presente experimento se le ingresaran imágenes con medida de (1,14), ya que al realizar la limpieza de los datos se eliminaron muchos períodos de tiempo sin información, lo cual provoca instantes en el tiempo que es imposible determinar el estado de los datos. Este problema es resuelto al observar los distintos sensores en solo 1 período de tiempo, ya que en ese instante es conocido el estado de los datos. Esta red tendrá ciertas características, las cuales son las siguientes:

- Número de capas: Este hiper-parámetro fue seleccionado por medio de realizar "ensayo y error", ya que se probaron distintos números de capas y se concluyo que el numero óptimo de capas para este modelo era 10 (al igual que con la red MLP).
- Padding: Todas las capas tendrán padding, de forma que al finalizar cada convolución la imagen de salida tenga el mismo tamaño que la de entrada.
- Número de épocas: Se testeara la red con 100 y 200 épocas, para así observar la influencia de este parámetro en la red.
- Función de activación: Se utilizara ReLU, ya que los datos están normalizados entre 0 y 1.
- Dropout: Se utiliza un valor de 20 %, es decir, ignoramos el 20 % de las neuronas.
- Batch Size: Se utilizan distintos batch sizes para así observar la influencia de este parámetro en la red. De igual modo, los batch size utilizados siempre sera divisor del numero total de datos ingresados.
- Al final de la red CNN habrá una red MLP, esto es debido a que las redes CNN se utilizan como extractor de características y no para realizar diagnóstico.

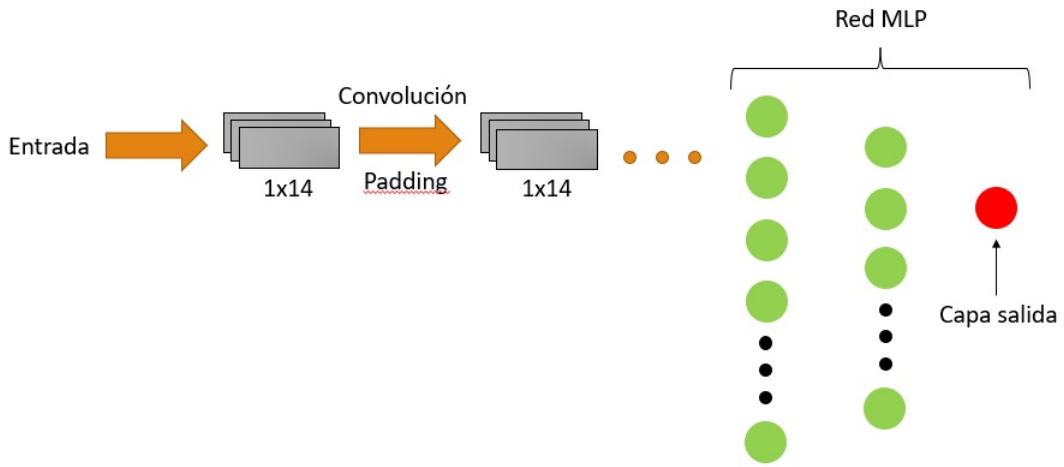

Figura 4.3: Esquema arquitectura utilizada

4.4.3. Experimento 3: SVM

El tercer experimento consiste en utilizar los datos en un SVM, de forma de comparar los 2 resultados de las redes anteriores con los resultados obtenidos de esta herramienta del Machine Learning. De este solo se obtiene la matriz de confusión, es decir, se compararán la cantidad de aciertos y fallos que tienen al momento de diagnosticar el estado del reboiler. Algunas características del SVM a utilizar son las siguientes:

- Función perdida: Se utiliza la función hinge, ya que esta es la utilizada para encontrar el máximo margen de clasificación.
- Cantidad de iteraciones: Se utilizan 1.000.000 iteraciones, esto en la variable max_iter.
- Kernel: Se utiliza LinearSVC, ya que con esta es posible obtener un hiper-plano que separe datos lineales.

Capítulo 5

Resultados

En esta sección se observaran los resultados obtenidos de los distintos experimentos, además de sus respectivos análisis.

5.1. Experimento 1: Red MLP

5.1.1. 100 épocas

5.1.1.1. Batch Size de 157

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9760
Val. acc.	0,9793
Test acc.	0,9767
Loss	0,0555
Val. loss	0,0591
Test loss	0,0677

Tabla 5.1: Valores de accuracy y loss batch 157

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.733	26
	Negative	134	3.976

Tabla 5.2: Matriz de confusión batch 157

- **Matriz de confusión normalizada**

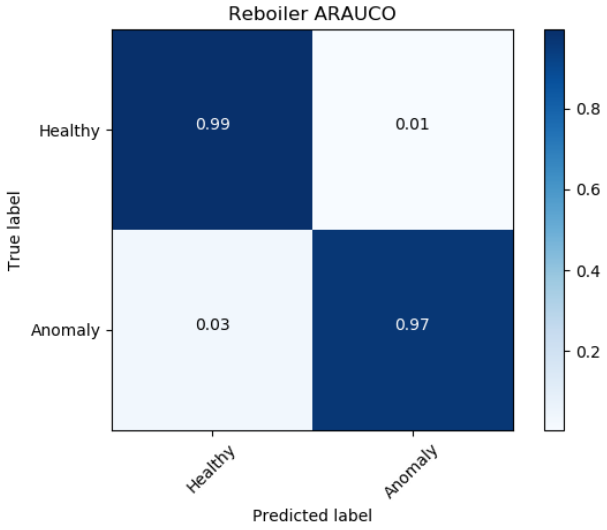

Figura 5.1: Matriz de confusión normalizada batch 157

- **Gráfico perdida vs épocas**

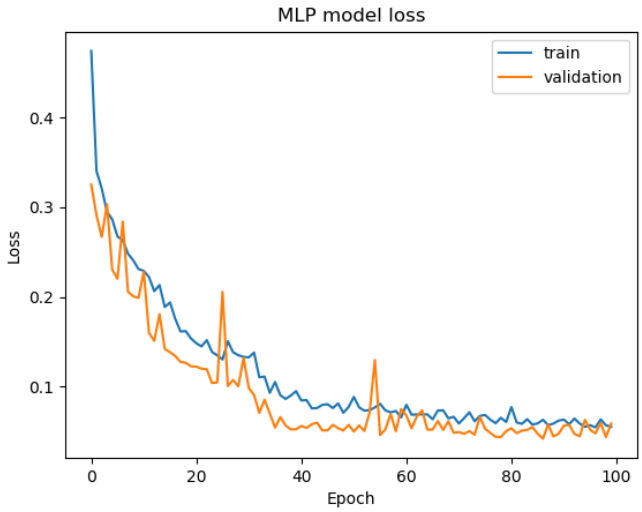

Figura 5.2: Gráfico de perdida batch 157

- **Gráfico accuracy vs épocas**

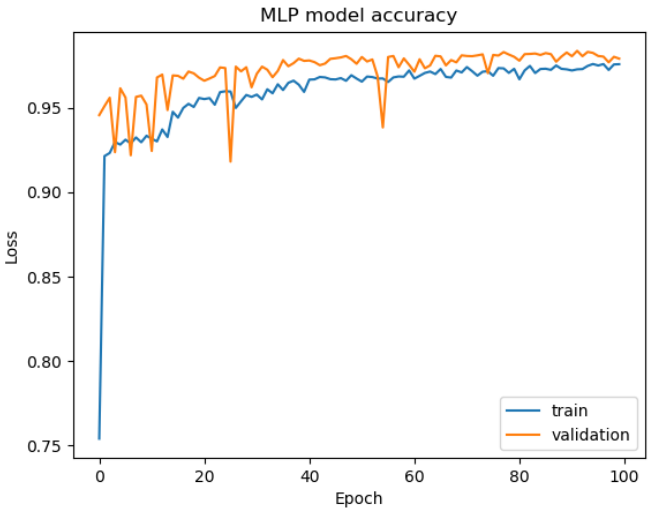

Figura 5.3: Gráfico de accuracy batch 157

De los resultados obtenidos, se puede observar valores de accuracy superiores al 97 %, lo cual indica un alto nivel de diagnostico del modelo. Con respecto a la tabla 5.2 y la figura 5.1, se puede observar que este modelo presenta mas problemas con los falsos positivos, de igual modo, son inferiores al 4 %, lo cual es un bajo valor. Con respecto a la figura 5.2 se puede observar que el modelo va disminuyendo su perdida rápidamente hasta la época 50, donde comienza a estabilizarse y por el lado de la figura 5.3, se observa que se obtiene un accuracy cercano al 90 % en las primeras épocas, pero este aumenta su valor hasta aproximadamente la época 50 donde nuevamente se comienza a estabilizar.

5.1.1.2. Batch Size de 175

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9772
Val. acc.	0,9802
Test acc.	0,9767
Loss	0,0518
Val. loss	0,0585
Test loss	0,0689

Tabla 5.3: Valores de accuracy y loss batch 175

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.727	32
	Negative	128	3.882

Tabla 5.4: Matriz de confusión batch 175

- **Matriz de confusión normalizada**

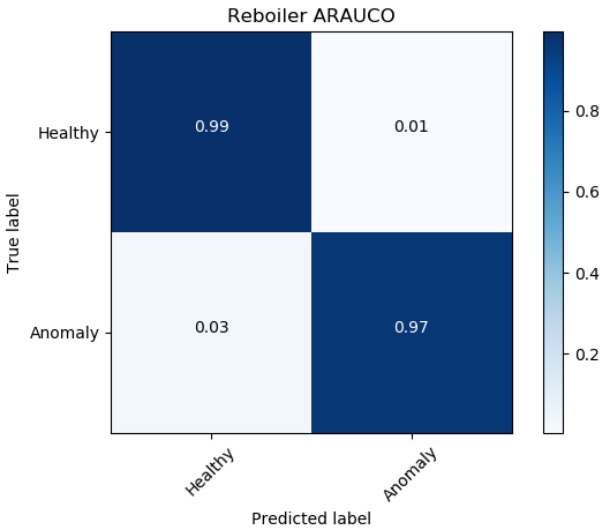

Figura 5.4: Matriz de confusión normalizada batch 175

- **Gráfico perdida vs épocas**

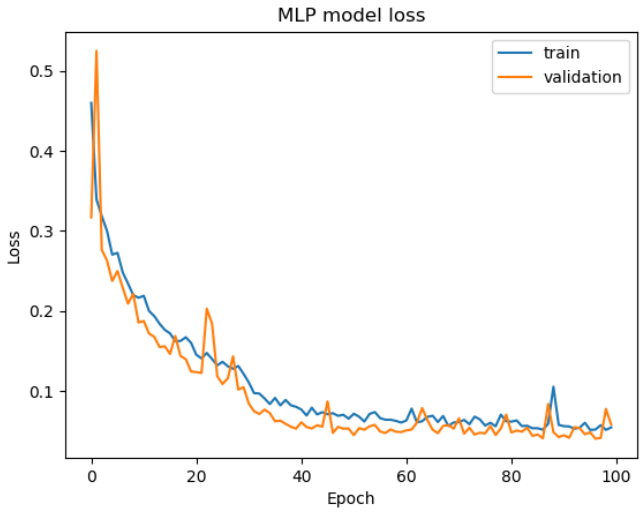

Figura 5.5: Gráfico de perdida batch 175

- **Gráfico accuracy vs épocas**

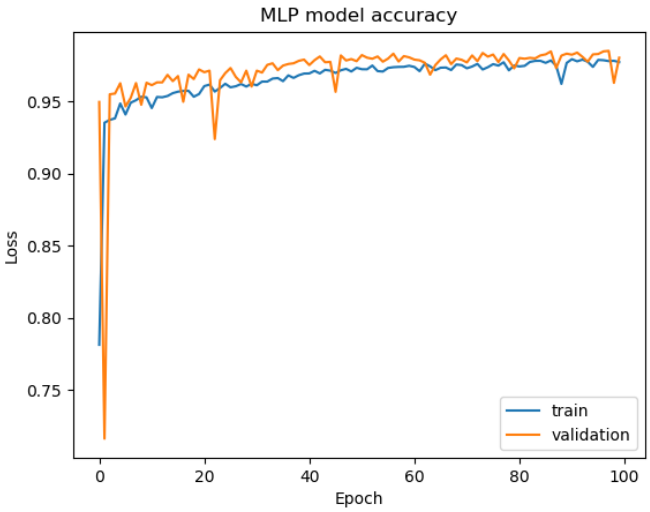

Figura 5.6: Gráfico de accuracy batch 175

De estos resultados se puede observar valores similares al caso anterior, ya que nuevamente se obtiene un accuracy superior al 97 %. Con respecto a la tabla 5.4 y figura 5.4, es posible observar que nuevamente la zona mas problemática son la de los falsos positivos, pero al igual que el caso anterior, es inferior al 4 %. La figura 5.5 muestra como desciende la perdida a lo largo de las épocas, llegando a valores inferiores a 10 % desde la época 40 en adelante. Por su parte, la figura 5.6 el accuracy aumenta rápidamente a 90 %en las primeras épocas, para luego ir aumentado hasta llegar a valores superiores al 95 %.

5.1.1.3. Batch Size de 785

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9712
Val. acc.	0,9754
Test acc.	0,9748
Loss	0,1130
Val. loss	0,1034
Test loss	0,1060

Tabla 5.5: Valores de accuracy y loss batch 785

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.735	24
	Negative	149	3.961

Tabla 5.6: Matriz de confusión batch 785

- **Matriz de confusión normalizada**

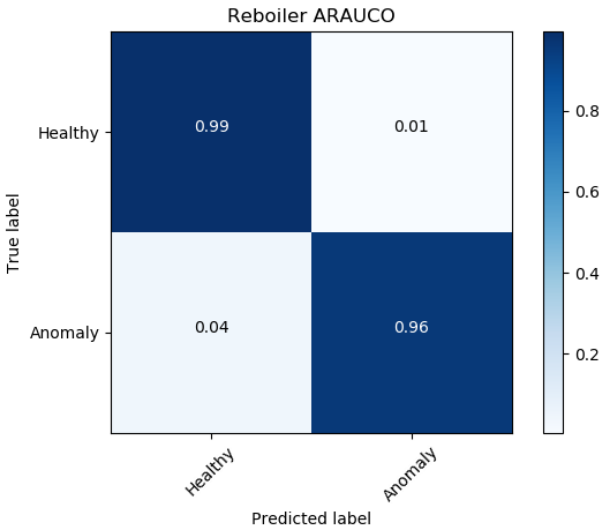

Figura 5.7: Matriz de confusión normalizada batch 785

- **Gráfico perdida vs épocas**

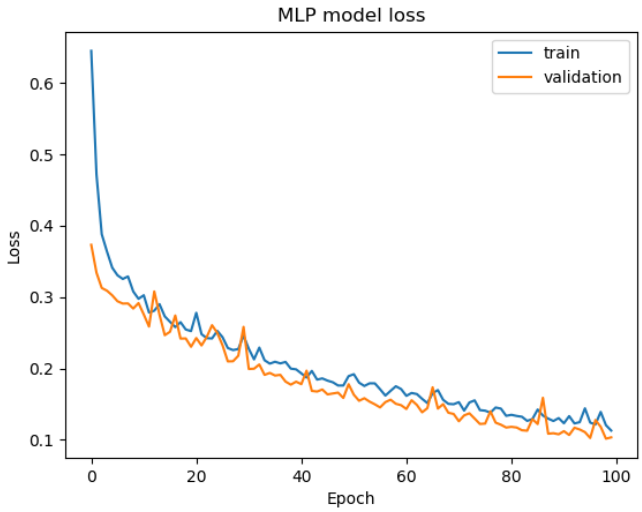

Figura 5.8: Gráfico de perdida batch 785

- **Gráfico accuracy vs épocas**

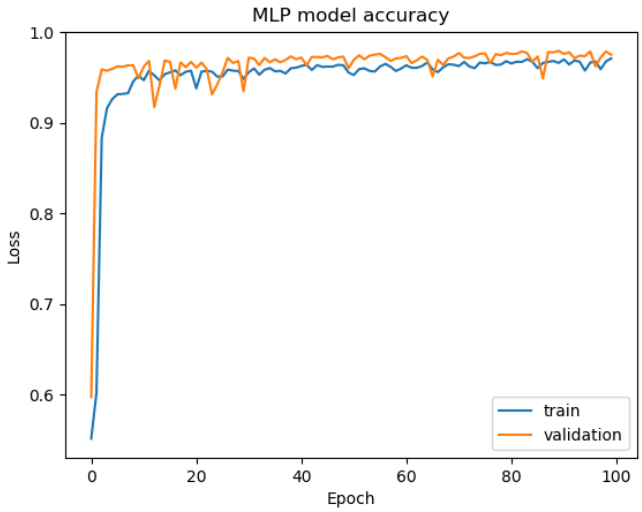

Figura 5.9: Gráfico de accuracy batch 785

Se observa nuevamente un accuracy cercano al 97 %. De la tabla 5.6 y la figura 5.7 es posible observar un leve aumento de los falsos positivos, ya que con esta configuración se obtienen valores de 4 %, aun así. Observando la figura 5.8 se observa como a lo largo de las 100 épocas la perdida disminuye hasta llegar a valores cercanos al 10 %. En la figura 5.9 se puede notar como el accuracy aumenta rápidamente en las primeras épocas y se mantiene practicamente estable a lo largo de las 100 épocas.

5.1.2. 200 épocas

5.1.2.1. Batch Size de 157

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9802
Val. acc.	0,9845
Test acc.	0,9791
Loss	0,0495
Val. loss	0,0397
Test loss	0,0505

Tabla 5.7: Valores de accuracy y loss batch 157

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.679	80
	Negative	63	4.047

Tabla 5.8: Matriz de confusión batch 157

- **Matriz de confusión normalizada**

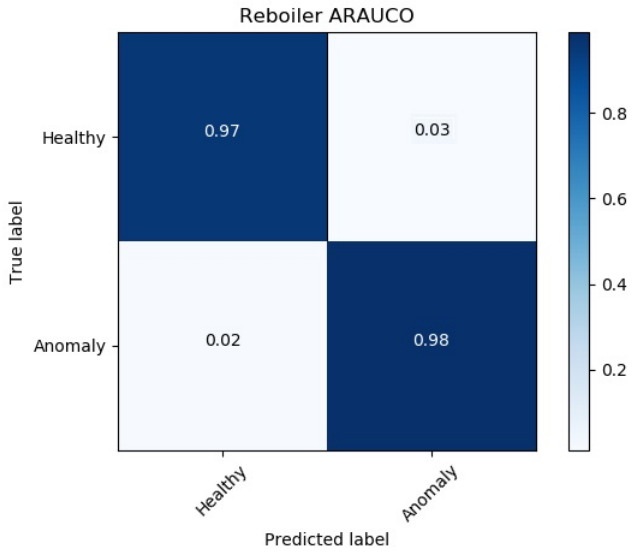

Figura 5.10: Matriz de confusión normalizada batch 157

- **Gráfico perdida vs épocas**

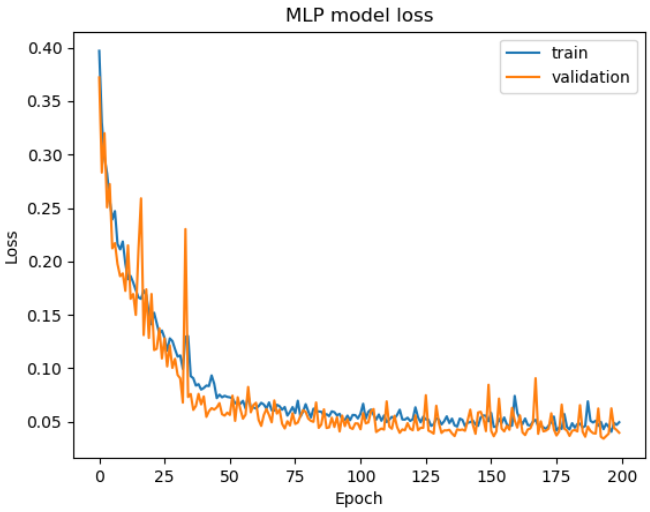

Figura 5.11: Gráfico de perdida batch 157

- **Gráfico accuracy vs épocas**

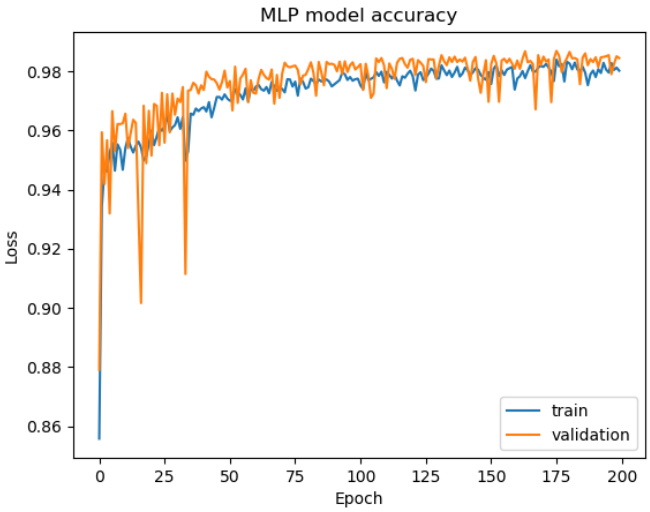

Figura 5.12: Gráfico de accuracy batch 157

De los resultados obtenidos, se puede observar valores de accuracy superiores al 97 %. Con respecto a la tabla 5.8 y la figura 5.10, se observa que los falsos positivos disminuyen a un 2 %, pero los falsos negativos aumentan a un 3 %. Con respecto a la figura 5.11 se observa una clara disminución de la perdida hasta la época 40 aproximadamente, desde la cual comienza a disminuir de forma mas lenta la perdida. La figura 5.12 muestra como el accuracy aumenta hasta la época 50 y desde dicha época comienza a estabilizarse.

5.1.2.2. Batch Size de 175

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9843
Val. acc.	0,9846
Test acc.	0,9734
Loss	0,0391
Val. loss	0,0474
Test loss	0,0576

Tabla 5.9: Valores de accuracy y loss batch 175

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.734	25
	Negative	84	4.021

Tabla 5.10: Matriz de confusión batch 175

- **Matriz de confusión normalizada**

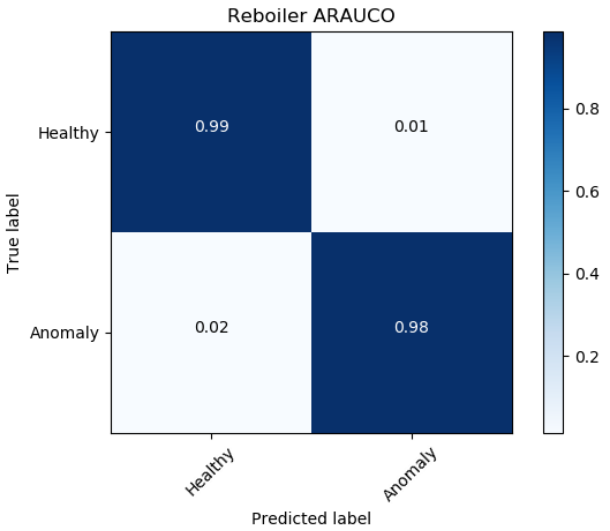

Figura 5.13: Matriz de confusión normalizada batch 175

- **Gráfico perdida vs épocas**

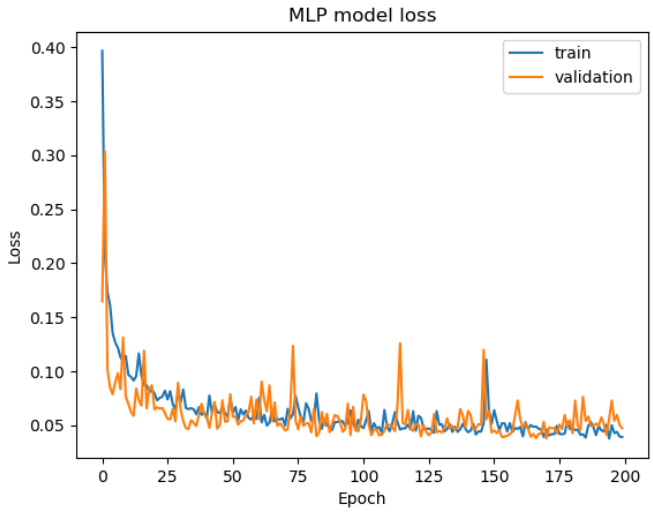

Figura 5.14: Gráfico de perdida batch 175

- **Gráfico accuracy vs épocas**

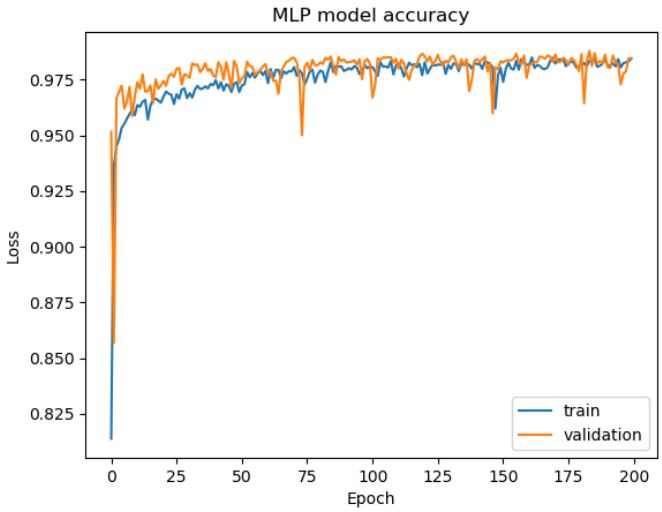

Figura 5.15: Gráfico de accuracy batch 175

Se observa una leve mejoría en el accuracy, alcanzando valores cercanos a 98 %. Por parte de la tabla 5.10 y la figura 5.13, es posible observar una leve disminución en los falsos positivos, alcanzando un valor de un 2 %. Con respecto a las figuras 5.14 y 5.15, se observan valores de perdida cercanos a 5 % desde las primeras épocas y accuracy superiores a 95 % en la misma zona.

5.1.2.3. Batch Size de 785

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9793
Val. acc.	0,9805
Test acc.	0,9803
Loss	0,0646
Val. loss	0,0632
Test loss	0,0617

Tabla 5.11: Valores de accuracy y loss batch 785

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.744	15
	Negative	120	3.990

Tabla 5.12: Matriz de confusión batch 785

- **Matriz de confusión normalizada**

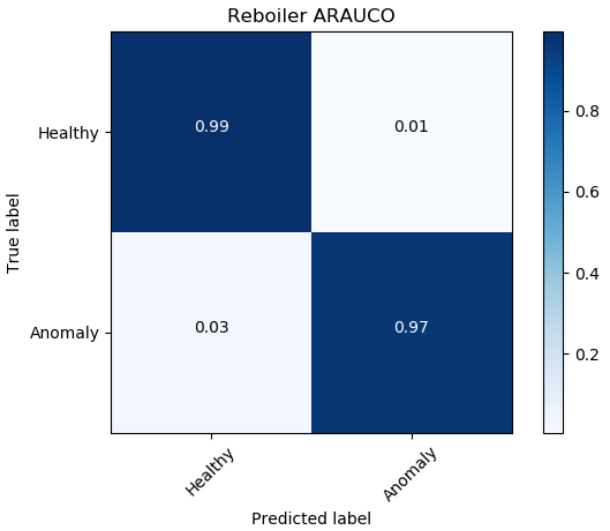

Figura 5.16: Matriz de confusión normalizada batch 785

- **Gráfico perdida vs épocas**

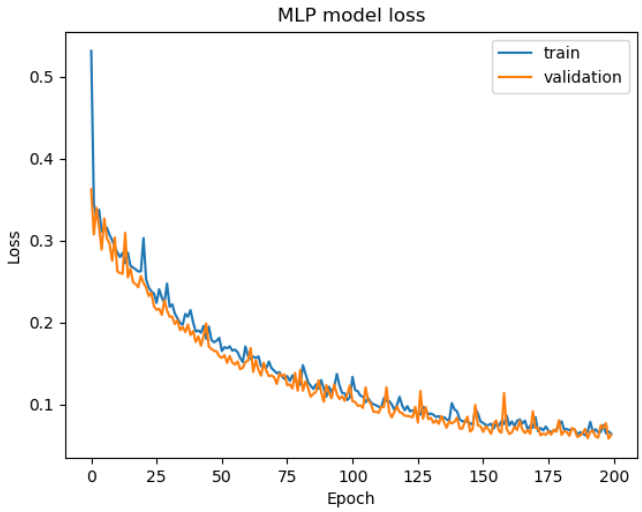

Figura 5.17: Gráfico de perdida batch 785

- **Gráfico accuracy vs épocas**

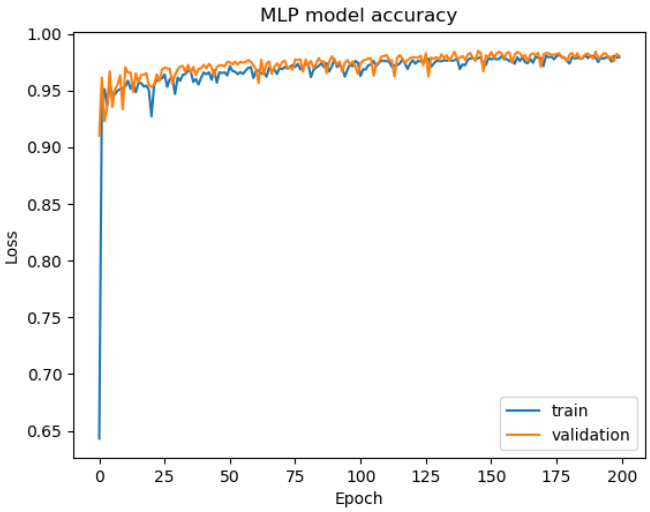

Figura 5.18: Gráfico de accuracy batch 785

Se observa un accuracy cercano al 98 %. De las tabla 5.12 y figura 5.16, se observa un leve aumento de los falsos positivos con respecto al caso anterior, alcanzando un 3 %. Con respecto a la figura 5.17 el grafico de perdida muestra como a lo largo de las 200 épocas la perdida cae desde valores superiores al 30 % a valores inferiores al 10 %. Por el lado de la figura 5.18, se logra notar como el accuracy es estable aproximadamente desde la época 25-30, obteniendo valores superiores al 95 %.

5.2. Red CNN

5.2.1. 100 épocas

5.2.1.1. Batch Size de 157

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9879
Val. acc.	0,9854
Test acc.	0,9839
Loss	0,0288
Val. loss	0,0362
Test loss	0,0368

Tabla 5.13: Valores de accuracy y loss batch 157

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.735	38
	Negative	64	4.032

Tabla 5.14: Matriz de confusión batch 157

- **Matriz de confusión normalizada**

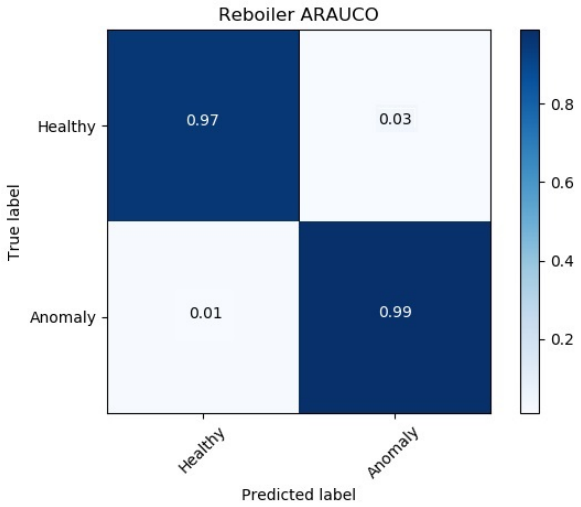

Figura 5.19: Matriz de confusión normalizada batch 157

- **Gráfico perdida vs épocas**

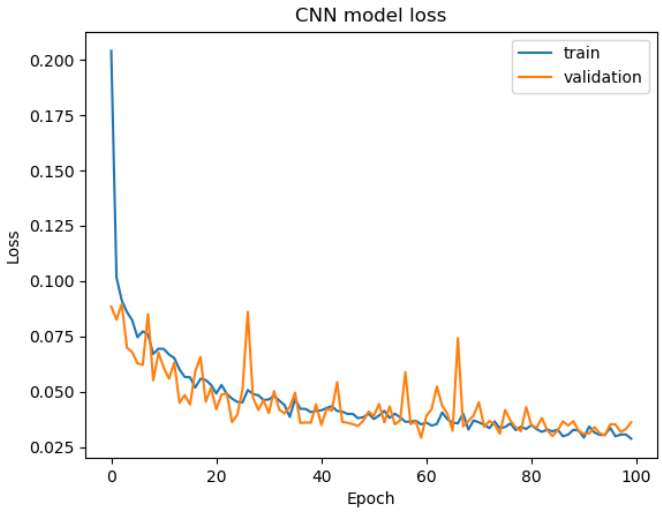

Figura 5.20: Gráfico de perdida batch 157

- **Gráfico accuracy vs épocas**

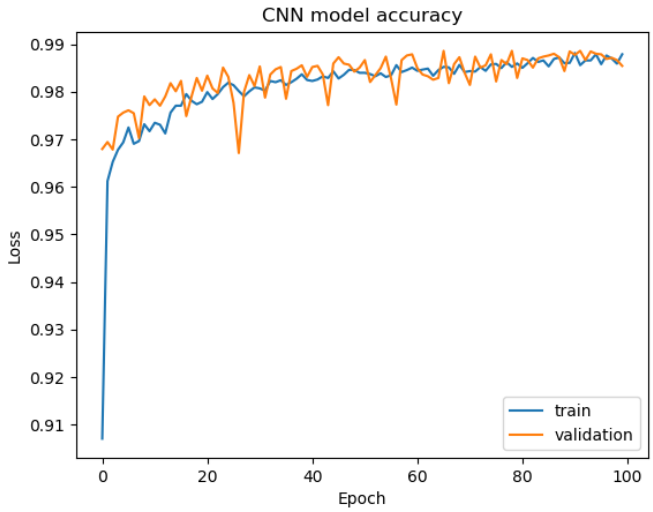

Figura 5.21: Gráfico de accuracy batch 157

Se observa un accuracy superior 98 %. De la tabla 5.14 y figura 5.19, es posible observar una gran estabilidad en la matriz de confusión, donde el valor mas fuera de rango son los falsos negativos con solo un 3 %. Por parte de la figura 5.20 se observa como en las primeras 60 épocas presenta una clara disminución de la perdida llegando a valores inferiores al 5 %. Por el lado de la figura 5.21, el accuracy aumenta hasta la época 50 hasta valores cercanos al 98 %, y desde dicha época el accuracy aumenta de forma mas estable hasta valores superiores a dicho porcentaje.

5.2.1.2. Batch Size de 175

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9865
Val. acc.	0,9848
Test acc.	0,9838
Loss	0,0295
Val. loss	0,0415
Test loss	0,0472

Tabla 5.15: Valores de accuracy y loss batch 175

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.729	30
	Negative	81	4.029

Tabla 5.16: Matriz de confusión batch 175

- **Matriz de confusión normalizada**

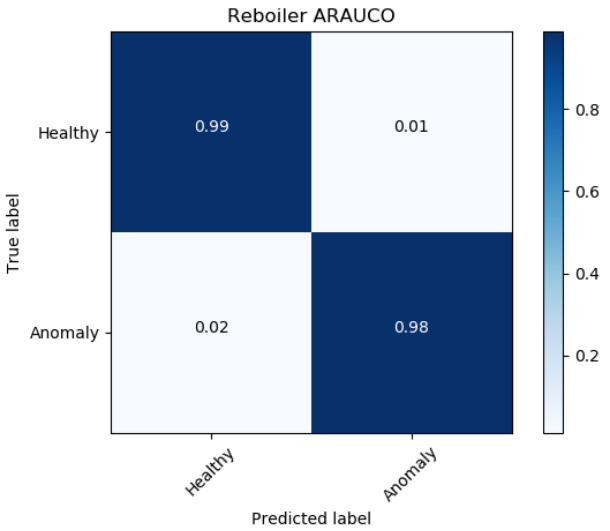

Figura 5.22: Matriz de confusión normalizada batch 175

- **Gráfico perdida vs épocas**

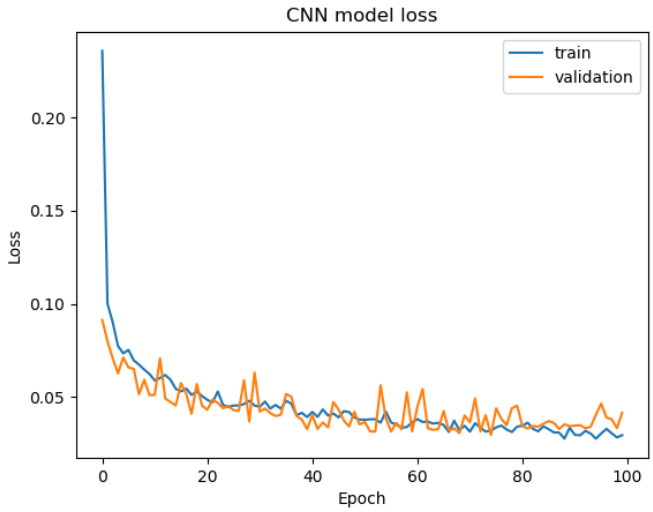

Figura 5.23: Gráfico de perdida batch 175

- **Gráfico accuracy vs épocas**

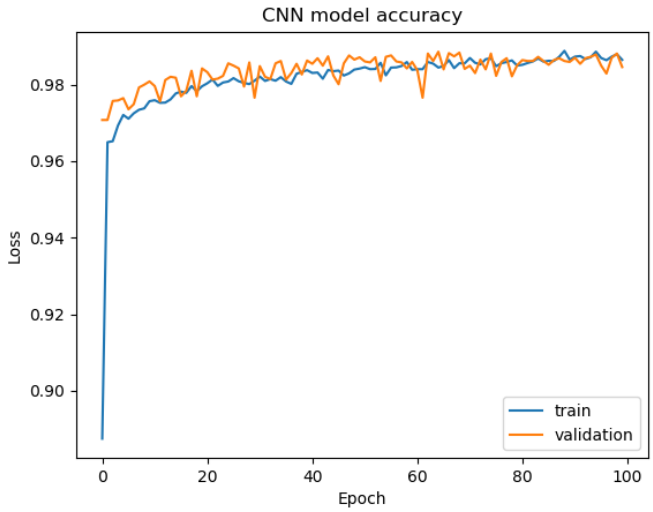

Figura 5.24: Gráfico de accuracy batch 175

Se observa un accuracy superior al 98 %. De la tabla 5.16 y figura 5.22, es posible observar que los falsos positivos tienen un valor del 2 %. Por parte de la figura 5.23 se observa como la perdida baja hasta 5 % en la época 35-40, desde la cual se estabiliza y comienza a descender de forma mas lenta. En la figura 5.24 se observa como en la época 20 se obtiene un accuracy cercano al 98 %, y desde dicha época, el accuracy aumenta de forma mas lenta.

5.2.1.3. Batch Size de 785

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9861
Val. acc.	0,9886
Test acc.	0,9855
Loss	0,0328
Val. loss	0,0301
Test loss	0,0327

Tabla 5.17: Valores de accuracy y loss batch 785

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.715	44
	Negative	55	4.055

Tabla 5.18: Matriz de confusión batch 785

- **Matriz de confusión normalizada**

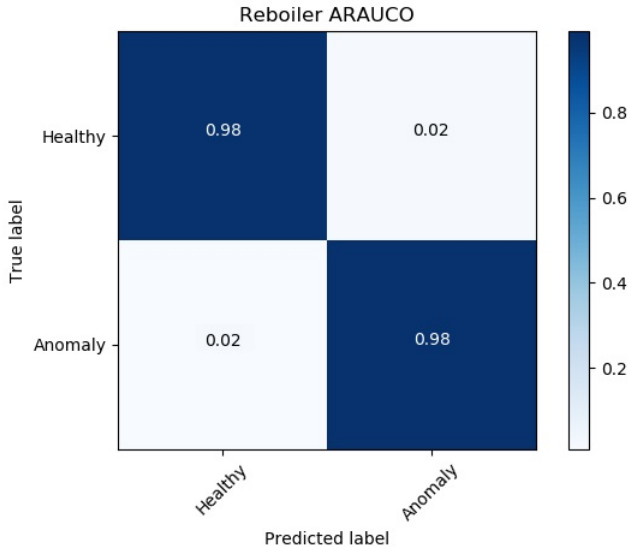

Figura 5.25: Matriz de confusión normalizada batch 785

- **Gráfico perdida vs épocas**

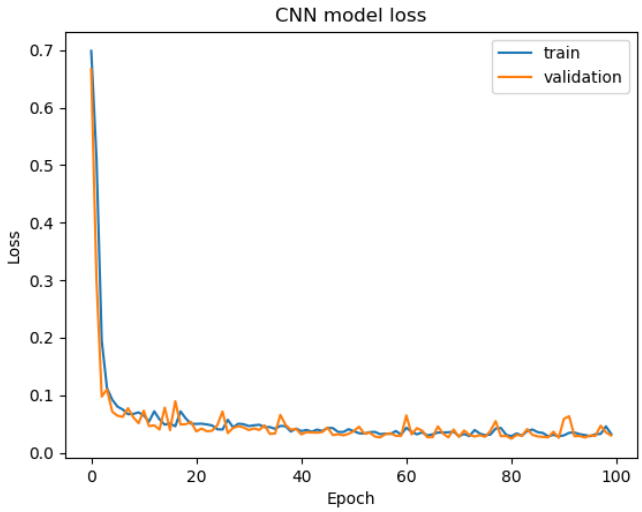

Figura 5.26: Gráfico de perdida batch 785

- **Gráfico accuracy vs épocas**

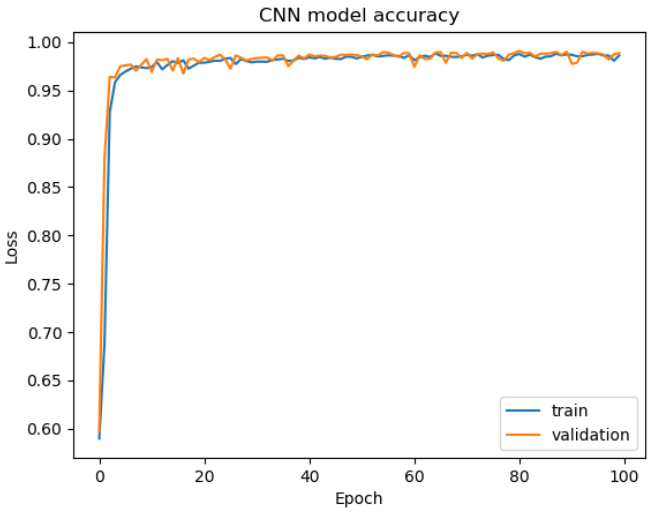

Figura 5.27: Gráfico de accuracy batch 785

Se observa un accuracy de 98 %. De la tabla 5.18 y figura 5.25, es posible observar que tanto los falsos positivos como negativos tienen un valor de un 2 %. Por parte de las figuras 5.26 y 5.27, se puede observar que en las épocas iniciales el modelo comienza a tener errores interiores al 10 % y un accuracy superior al 95 %.

5.2.2. 200 épocas

5.2.2.1. Batch Size de 157

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9915
Val. acc.	0,9879
Test acc.	0,9857
Loss	0,0205
Val. loss	0,0366
Test loss	0,0362

Tabla 5.19: Valores de accuracy y loss batch 157

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.707	52
	Negative	46	4.064

Tabla 5.20: Matriz de confusión batch 157

- **Matriz de confusión normalizada**

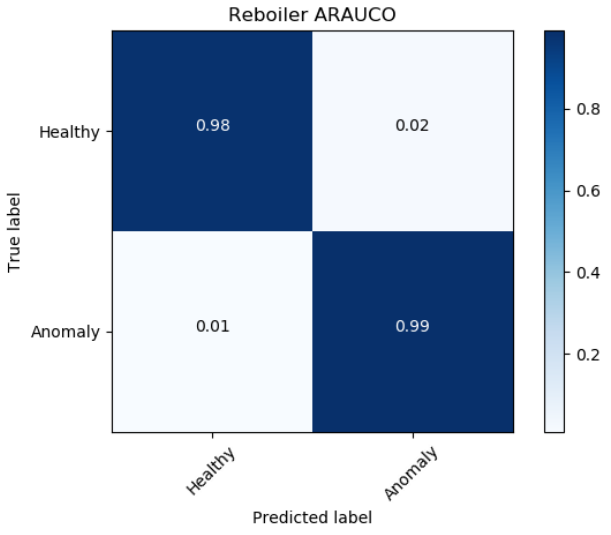

Figura 5.28: Matriz de confusión normalizada batch 157

- **Gráfico perdida vs épocas**

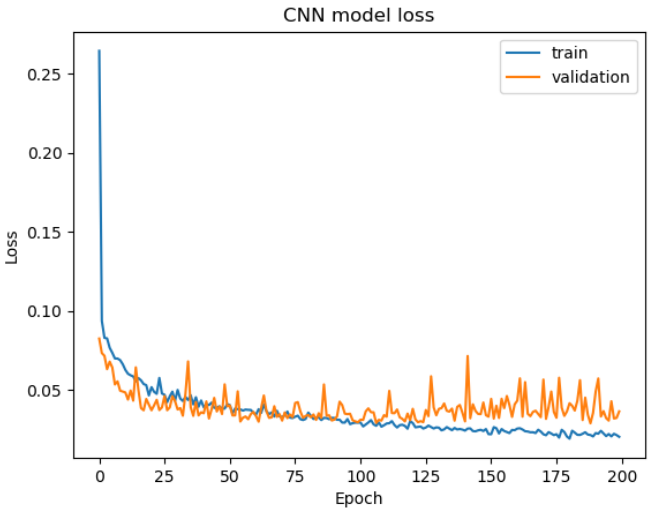

Figura 5.29: Gráfico de perdida batch 157

- **Gráfico accuracy vs épocas**

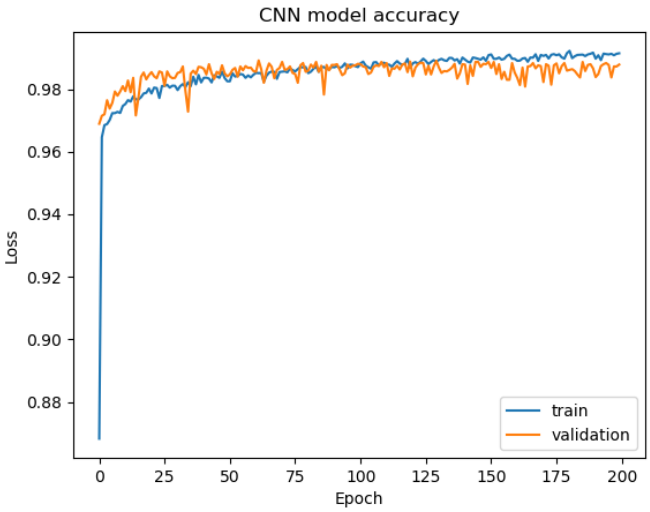

Figura 5.30: Gráfico de accuracy batch 157

Se observa un accuracy cercano al 99 %. De la tabla 5.20 y figura 5.28, se observa que falsos negativos tienen un valor de 2 %, lo cual en este caso supera a los falsos positivos. Por parte de las figuras 5.29 y 5.30, se puede observar que en las épocas iniciales el modelo comienza a tener errores interiores al 5 % y un accuracy superior al 96 %, también se logra observar que las curvas de perdida y accuracy comienzan a separarse desde la época 100.

5.2.2.2. Batch Size de 175

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9901
Val. acc.	0,9885
Test acc.	0,9887
Loss	0,0227
Val. loss	0,0295
Test loss	0,0290

Tabla 5.21: Valores de accuracy y loss batch 175

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.742	17
	Negative	60	4.050

Tabla 5.22: Matriz de confusión batch 175

- **Matriz de confusión normalizada**

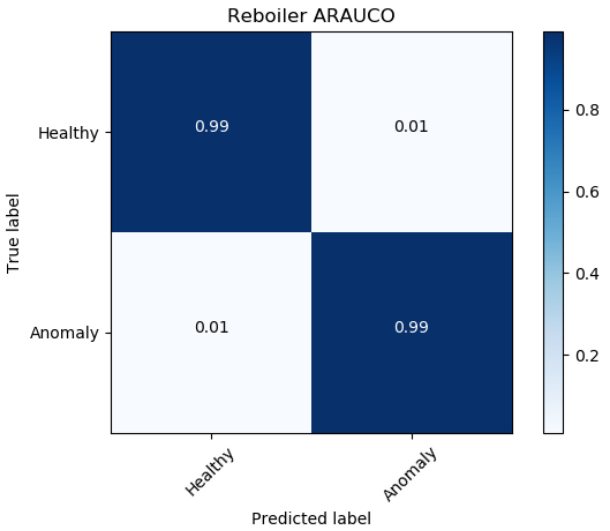

Figura 5.31: Matriz de confusión normalizada batch 175

- **Gráfico perdida vs épocas**

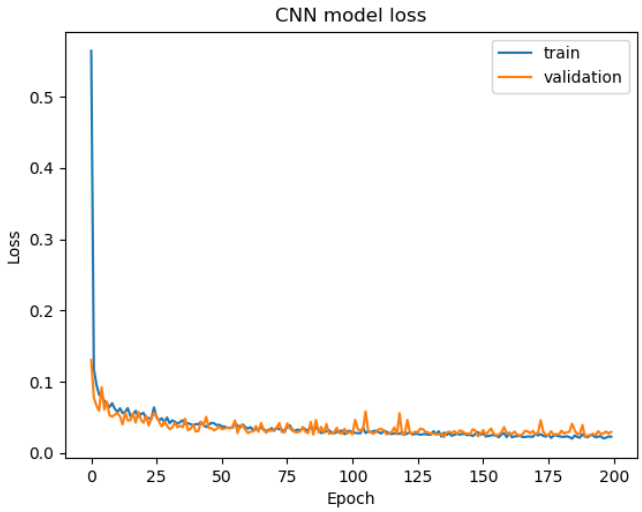

Figura 5.32: Gráfico de perdida batch 175

- **Gráfico accuracy vs épocas**

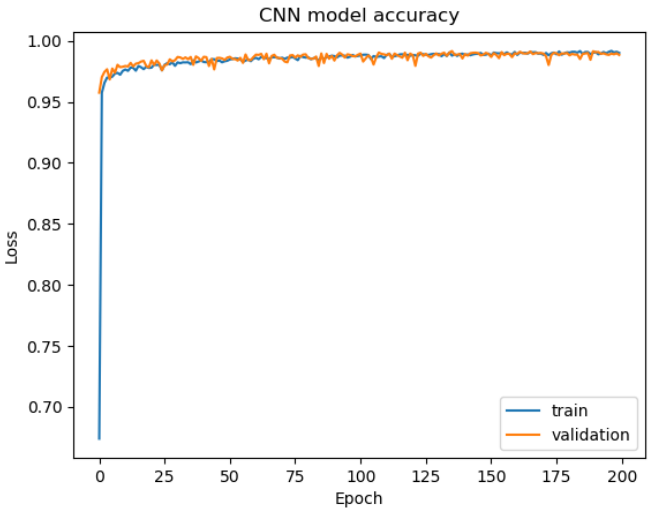

Figura 5.33: Gráfico de accuracy batch 175

Se observa un accuracy cercano a 99 %. De la tabla 5.22 y figura 5.31, se observa una matriz de confusión estable, donde tanto los falsos negativos como positivos llegan solo al 1 %. Por parte de las figuras 5.32 y 5.33, se puede observar que en las épocas iniciales el modelo comienza a tener errores interiores al 10 % y un accuracy superior al 95 %.

5.2.2.3. Batch Size de 785

- **Accuracy y Loss**

Indicador	Valor
Acc.	0,9908
Val. acc.	0,9884
Test acc.	0,9866
Loss	0,0216
Val. loss	0,0300
Test loss	0,0336

Tabla 5.23: Valores de accuracy y loss batch 785

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.697	62
	Negative	30	4.080

Tabla 5.24: Matriz de confusión batch 785

- **Matriz de confusión normalizada**

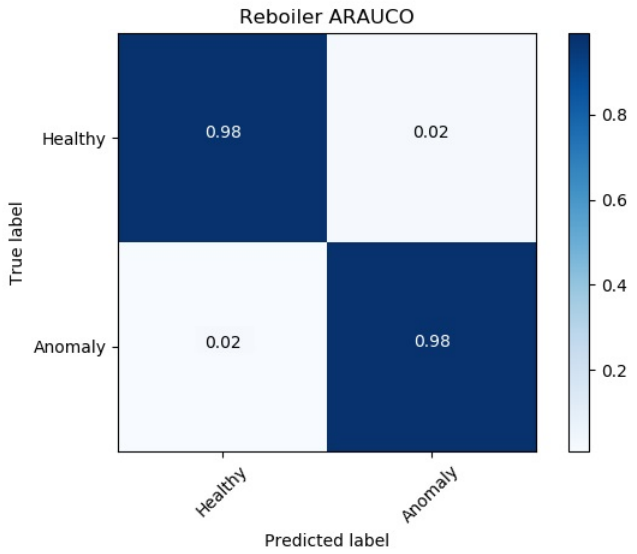

Figura 5.34: Matriz de confusión normalizada batch 785

- **Gráfico perdida vs épocas**

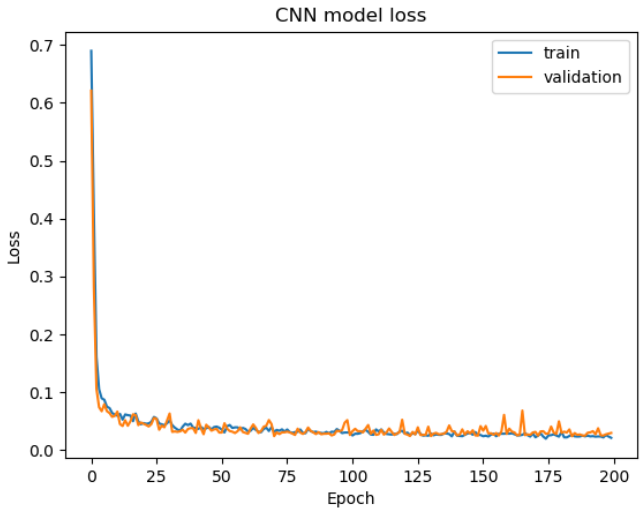

Figura 5.35: Gráfico de perdida batch 785

- **Gráfico accuracy vs épocas**

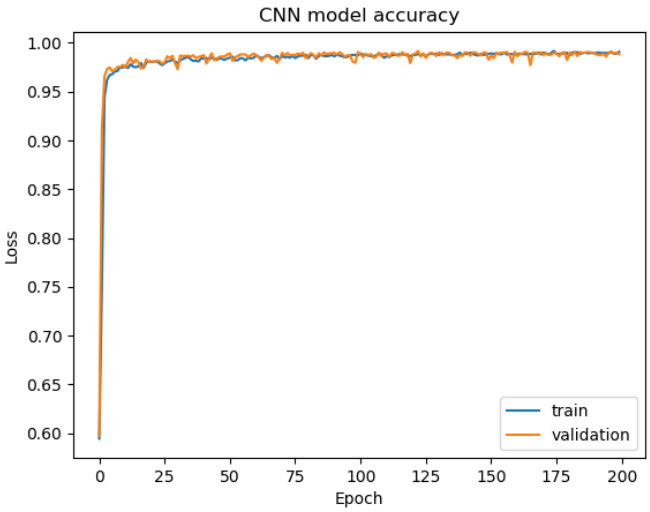

Figura 5.36: Gráfico de accuracy batch 785

Se observa un accuracy cercanos a 99 %. De la tabla 5.24 y figura 5.34, se observa que tanto los falsos negativos como positivos es de 2 %. Por parte de las figuras 5.35 y 5.36, se puede observar que en las épocas iniciales el modelo comienza a tener errores interiores al 10 % y un accuracy superior al 95 %.

5.3. SVM

Es posible observar resultados positivos tanto para MLP como CNN, pero estas redes llegan rápidamente a valores altos de accuracy y bajos de perdida. Por lo cual, para conocer mas en profundidad las características de los datos se procede a comparar un SVM. Para ello se obtuvo la matriz de confusión de esta, la cual se observa a continuación:

- **Matriz de confusión**

		Predicted	
		Positive	Negative
True	Positive	2.625	121
	Negative	179	3.944

Tabla 5.25: Matriz de confusión SVM

- **Matriz de confusión normalizada**

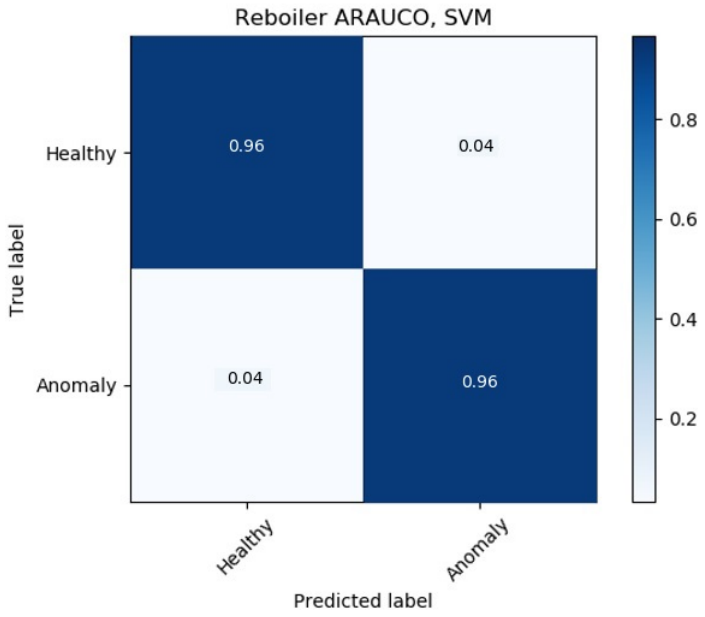

Figura 5.37: Matriz de confusión normalizada SVM

5.4. Tiempos de procesamiento

Modelo	Épocas	Tiempo procesamiento [s]
MLP	100	200
MLP	200	400
CNN	100	300
CNN	200	600
SVM	-	20

Tabla 5.26: Distintos tiempos de procesamiento

5.5. Resumen de resultados

100 épocas							
Modelo	Batch Size	Acc.	Val. Acc.	Test Acc.	Loss	Val. loss	Test loss
MLP	157	0,9760	0,9793	0,9767	0,0555	0,0591	0,0677
	175	0,9772	0,9802	0,9767	0,0518	0,0585	0,0689
	785	0,9712	0,9754	0,9748	0,1130	0,1034	0,1060
CNN	157	0,9879	0,9854	0,9839	0,0288	0,0362	0,0368
	175	0,9865	0,9846	0,9838	0,0295	0,0415	0,0472
	785	0,9861	0,9886	0,9855	0,0328	0,0301	0,0327

Tabla 5.27: Comparación de los modelos 100 épocas

200 épocas							
Modelo	Batch Size	Acc.	Val. Acc.	Test Acc.	Loss	Val. loss	Test loss
MLP	157	0,9802	0,9845	0,9791	0,0495	0,0397	0,0505
	175	0,9843	0,9846	0,9734	0,0391	0,0474	0,0576
	785	0,9793	0,9805	0,9803	0,0646	0,0632	0,0617
CNN	157	0,9915	0,9879	0,9857	0,0205	0,0366	0,0362
	175	0,9901	0,9885	0,9887	0,0227	0,0295	0,0295
	785	0,9908	0,9884	0,9866	0,0216	0,0300	0,0336

Tabla 5.28: Comparación de los modelos 200 épocas

Capítulo 6

Análisis de resultados

De los resultados obtenidos a través de la utilización de los modelos anteriormente nombrados, se pueden realizar las siguientes observaciones.

Se observa que los 2 tipos de redes (MLP y CNN) se obtienen resultados esperados, ya que se obtienen valores superiores al 97 % de accuracy en prácticamente todos los casos, lo cual indica la viabilidad de la utilización de ambos modelos en el diagnóstico de estado de salud.

Con respecto a los resultados de la red MLP con 100 épocas, se observa que el mayor problema son los falsos positivos, donde en el batch size de 785 alcanza un valor de un 4 %, lo cual es un valor aceptable de error. Por otro lado, tanto el batch size de 157 como el de 175 arrojan valores un poco mejores, alcanzando un 3 % de falsos positivos. Con respecto a las gráficas, se observa que el de perdida en todos los casos tiene un comportamiento similar, teniendo en una zona un gran descenso de la perdida y luego estabilizándose, por otro lado, las de accuracy se observa que en las primeras épocas aumenta rápidamente a valores superiores al 90 %.

La red MLP con 200 épocas tiene menos falsos positivos en general, llegando a un valor máximo de 3 % en la red con batch size de 785. A diferencia del caso anterior, aquí aumenta levemente los falsos negativos, llegando a valores de un 3 % en el caso del batch size de 157. Es importante recordar que los falsos negativos son menos problemáticos que los falsos positivos, ya que los falsos positivos son muestras incorrectamente clasificadas como positivas, es decir, que al momento de identificar el estado de alguna pieza, el falso positivo indica que dicha pieza esta funcional cuando en realidad esta fallando. Por otro lado, los falsos negativos son conocidas como 'falsa alarma', ya que son piezas identificadas como que fallan, pero en realidad están funcionales.

Por el lado de las gráficas de la red MLP de 200 épocas, estas son muy similares al caso anterior, solo que como en este caso son 200 épocas en vez de 100. Aun así, es importante destacar la figura 5.14 (batch size 175), la cual muestra una estabilidad en la perdida desde una época mas temprana.

Por el lado de la red CNN de 100 épocas, se observa una disminución en los falsos positivos, alcanzando valores máximos de un 2 %. También es importante destacar que los falsos negativos aumentan levemente llegando a valores máximos de 3 %. Con respecto a las gráficas de perdida, se observa que la red con batch size de 157 disminuye su perdida rápidamente hasta cierto punto donde se estabiliza, la red de 175 obtiene dicho punto de estabilización en una época mas temprana, y la de 785 esta estable desde épocas muy iniciales. Las gráficas de accuracy presentan un comportamiento similar, donde la de batch size 157 se estabiliza en una época mas tardía que los otros 2.

La red CNN de 200 épocas posee problemas de sobre-entrenamiento, esto a causa de que en las gráficas de perdida, las curva de entrenamiento comienza a ser inferior a la curva de validación, esto se observa de forma clara en el batch size de 157. También se observa que todas las curvas llegan a errores y accuracy estables en muy pocas épocas. Con respecto a los falsos positivos, estos en los 3 casos alcanzan solo un 1 %, y los falsos negativos llegan a un máximo de 2 %.

En las curvas MLP se observa el comportamiento esperado, donde a lo largo de las épocas la red va disminuyendo la perdida y aumentando su accuracy, de igual modo, en las primeras épocas el accuracy en este tipo de redes es cercano al 90 %, lo cual indica un rápido aprendizaje de este tipo de red. Por parte de las curvas en la red CNN, la mayoría de ellas presenta un comportamiento donde en las primeras épocas hay una gran aprendizaje, tanto en accuracy como en perdida, ya que en las primeras épocas se obtienen valores altos de accuracy y bajos de perdida. De igual modo, ambas redes realizan un diagnóstico del estado de los reboilers, llegando a errores inferiores al 4 % en todos los casos. También es importante comparar

El gran aprendizaje en las primeras épocas en ambas redes indica que los datos recibidos tienen un comportamiento demasiado simple para la utilización de redes neuronales, por lo cual se proceder a comparar los resultados obtenidos con técnicas del Machine Learning, en este caso una SVM. De esta comparación se puede observar de la tabla 5.25 y la figura 5.37, que se obtienen valores de al 96 % de precisión tanto en verdaderos positivos como negativos, esto indica su alta capacidad de diagnóstico.

Capítulo 7

Conclusiones

En base al trabajo realizado, resultados obtenidos y sus respectivos análisis, se procede a concluir.

En primer lugar es importante destacar que los objetivos, tanto general como especifico, se cumplen. Con respecto al objetivo general, se desarrolló un algoritmo, en este caso un modelo de redes neuronales, el cual es capaz de diagnosticar el estado de salud de los reboilers de bombas industriales. Por el lado de los objetivos específicos, es posible observar que se cumple cada uno de ellos. Inicialmente los datos brutos son estudiados y procesados, de forma de obtener un conjunto de datos aptos para su utilización en un modelo. Posteriormente se realiza un modelo de aprendizaje profundo, el cual, no sufre sobre-ajuste (overffiting). Del modelo realizado se obtienen diversos resultados, los cuales son analizados y se utilizan para observar que combinación de parámetros es la mas apta para recibir los datos.

Con respecto a los datos recibidos, es posible observar que poseían muchos espacios temporales sin información, sensores apagados y además carecían de etiqueta. Con esto se pudo observar que la calidad de los datos no era la óptima, por lo cual, los datos fueron tratados, de forma de obtener una matriz apta para el ingreso al modelo diseñado.

Por el lado del modelo, es importante destacar que se utilizaron 2 tipos de redes neuronales, CNN y MLP, y que en ambas se obtuvieron valores superiores al 97 % de accuracy, y observando los gráficos de perdida y los valores de validación, se concluye que en ninguno de los casos estudiados el modelo sufre de overfitting.

Se observa que el modelo tiene un mejor accuracy es el que utiliza una red CNN, con un batch size de 157 y 200 épocas. El mejor accuracy de una red MLP es el que tiene 200 épocas y un batch size 785.

También se concluye la simplicidad de los datos, ya que el SVM logra obtener resultados similares a las redes neuronales, solo teniendo un 4 % de falsos negativos y positivos.

Otro punto importante a destacar es el tiempo de procesamiento, ya que, comparando ambas redes con la misma cantidad de épocas, la red MLP fue 1,5 veces mas rápida que la red CNN. Además se observa que el SVM procesa 10 veces mas rápido que que la red MLP

de 100 épocas.

Finalmente es importante destacar que la elección de la el método a utilizar para el diagnóstico influye directamente en los resultados obtenidos, por lo cual a la hora de seleccionar una de ellas se debe estudiar cual es la prioridad de la empresa, mas precisión o menor tiempo, ya que a la hora de buscar la mejor precisión se selecciona la red CNN o MLP, ya que ambas obtuvieron valores muy similares, pero si la empresa requiere de resultados rápidos y que de igual forma sean confiables es recomendable utilizar el método SVM.

Capítulo 8

Bibliografía

[1] LÓPEZ, E. 2018. "Deep Neural Networks for fault diagnostics and prognostics". [Diapositivas]. Facultad de ciencias físicas y matemáticas, Universidad de Chile. Texto en inglés. 149p.

[2] GOODFELLOW, I., BENGIO, Y., COURVILLE, A. 2016. "Deep Learning". MIT Press <http://www.deeplearningbook.org> [Consulta: 25 noviembre 2018]

[3] BAGNATO, J. 12 de diciembre de 2017. "Qué es overfitting y underfitting y cómo solucionarlo". [En línea] <http://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/> [Consulta: 28 noviembre 2018].

[4] XIAOHAN2012. 28 de noviembre 2011. "What is the difference between test set and validation set?". Cross Validated. [En línea] <https://stats.stackexchange.com/questions/19048/what-is-the-difference-between-test-set-and-validation-set> [Consulta: 28 noviembre 2018]

[5] NAZZAL J., EL-EMARY I., NAJIM S. "Multilayer Perceptron Neural Network (MLPs) For Analyzing the Properties of Jordan Oil Shale", World Applied Sciences Journal. 2008.

[6] BROWNLEE, J. 18 de noviembre 2016. "What is a Confusion Matrix in Machine Learning". [En línea]. <https://machinelearningmastery.com/confusion-matrix-machine-learning/> [Consulta: 30 noviembre 2018]

[7] Advanced Tech Computing Group UTPL. 31 de agosto de 2007. "Elementos básicos de una red neuronal artificial". [En línea]. <https://advancedtech.wordpress.com/2007/08/31/elementos-basicos-de-una-red-neuronal-artificial/> [Consulta: 30 noviembre 2018]

[8] SHARMA, A. 30 de marzo 2017. Ünderstanding Activation Functions in Neural Networks. The Theory Of Everything". [En línea]. <https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0> [Consulta: 01 diciembre 2018]

[9] PRABHU. 04 de marzo 2018. Ünderstanding of Convolutional Neural Network (CNN)—Deep Learning. The Theory Of Everything". [En línea]. <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148> [Consulta: 3 diciembre 2018]

[10] DELMAR, F. MINETTO, R. TOMOYUKY, B. Çonvolutional Neural Networks for License Plate Detection in Images". Federal University of Technology — Parana, Brazil. Septiembre, 2017.

[11] REPPEL, E. 23 de enero 2017. "Visualizing parts of Convolutional Neural Networks using Keras and Cats". [En línea]. <https://hackernoon.com/visualizing-parts-of-convolutional-neural-networks-using-keras-and-cats-5cc01b214e59> [Consulta: 4 diciembre 2018]

[12] COMPSI. 19 de febrero 2018. "Twitter sentiment Analysis using Combined LSTM-CNN models". [En línea]. <http://konukoii.com/blog/2018/02/19/twitter-sentiment-analysis-using-combined-lstm-cnn-models/> [Consulta: 23 diciembre 2018]

[13] MORENO, R. Marzo 2016. "Deep Learning: qué es y por qué va a ser una tecnología clave en el futuro de la inteligencia artificial". [En línea]. <https://www.xataka.com/robotica-e-ia/deep-learning-que-es-y-por-que-va-a-ser-una-tecnologia-clave-en-el-futuro-de-la-inteligencia-artificial> [Consulta: 25 marzo 2019]

[14] HUANG, G. 2 de Marzo 2003. "Learning Capability and Storage Capacity of Two-Hidden-Layer Feedforward Networks". IEEE Transactions on Neuronal Network, vol. 14.

[15] SHIBATA, K.; IKEDA, Y. ”Effect of number of hidden neurons on learning in large-scale layered neural networks". Fukuoka International Congress Center, Japón. Agosto 2019.

[16] HUNTER, D.; YU, H. "Selection of Proper Neural Network Sizes and Architectures—A Comparative Study". IEEE Transactions on Industrial Informatics. Febrero 2012.

[17] ALLIBHAI, E. "Building a Convolutional Neural Network (CNN) in Keras". [En línea]. <https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5> [Consulta: 15 mayo 2019]

[18] ZHANG A., LIPTON Z., LI M., SMOLA A. "Dive into Deep Learning". [En línea] <https://www.d2l.ai/index.html> [Consulta: 1 septiembre 2019]

[19] GAJARDO, C., URRUTIA, G. 2019. [Diapositivas] "Implementación de Modelo de Big Data para la identificación temprana de fallas en Reboiler". Arauco. 33p

[20] KAPOOR, A. 2019. "Deep Learning vs. Machine Learning: A Simple Explanation". Hackernoon <https://hackernoon.com/deep-learning-vs-machine-learning-a-simple-explanation-47405b3eef08> [Consulta: 03 octubre 2019]

[21] SHASHANK, R. .A guide to an efficient way to build neural network architectures- Part II: Hyper-parameter selection and tuning for Convolutional Neural Networks using Hyperas on Fashion-MNIST". [En línea]. <https://towardsdatascience.com/a-guide-to-an-efficient-way-to-build-neural-network-architectures-part-ii-hyper-parameter-42efca01e5d7> [Consulta: 09 octubre 2019]

[22] JOLLIFFE, IT. 2002. "Principal component analysis". Segunda edición. Springer. 487p.

[23] WOWEZKO, I. "Density-based clustering with DBSCAN and OPTICS". Institute of Technology Blanchardstown. 2013.

[24] GANDHI, R. "Support Vector Machine — Introduction to Machine Learning Algorithms". [En línea]. <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47> [Consulta: 12 octubre 2019]

[25] GÉRON, A. "Hands-On Machine Learning with Scikit-Learn & TensorFlow". 1° edición. O'Reilly. 2017. 549p.

[26] COFRÉ, S. 2017. "Modelo de detección de fallas y faltas para sistema sistema neumático de turbinas de aviones Boeing 767 a través de Machine Learning". Memoria Ingeniero Civil Mecánico. Santiago, Universidad de Chile, Facultad de Ciencias Físicas y Matemáticas. 42p.

Capítulo 9

Anexos

9.1. Parámetros y rangos de operación del reboiler

Valores y rangos de operación		
	Valor	Rango
Conductividad de condensado	5 $\frac{\mu S}{cm}$	0-10 $\frac{\mu S}{cm}$
Nivel de tanque de condensado	50 %	40-60 %
Presión del vapor al hervidor	0 bar	-0.3 - 0.7 bar
Temperatura del vapor al hervidor	100 °C	80 - 110 °C

Tabla 9.1: Valores y rangos de operación reboiler

9.2. Código pre-procesamiento de datos

```
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans #si es que
from sklearn.cluster import DBSCAN #si es que
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LogisticRegression

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import keras
from keras.models import Sequential
```

```
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.utils import np_utils

#%%abrir datos con etiqueta

data_et = pd.read_excel('Datos_et.xlsx', header = 0)

et_arauco1 = np.array(data_et['Estado 2'])


et_arauco = []
for a in range((et_arauco1.shape[0])):
    if a>39719:
        et_arauco.append(et_arauco1[a])
et_arauco = np.asarray(et_arauco)
#et_arauco1 = et_arauco1.tolist()
relleno = np.ones(49526)*10

et_arauco = np.concatenate((relleno, et_arauco), axis = 0)
et_arauco = pd.DataFrame(et_arauco)

#%%
data= pd.read_excel('DATOS1.xlsx', header = 0) #sin etiqueta

# Eliminando tiempo y agregando etiqueta

data.drop(["Fecha"], axis = 1, inplace = True)
data.drop([0], axis = 0, inplace = True)
el = data.notnull()*1

suma_el = el.sum()
indices_datos = []
for z in range(len(suma_el)):
    if suma_el[z] >= el.shape[0]*0.75:
        indices_datos.append(z)

datos_up = data.iloc[:, indices_datos] #falta borrar hacia el lado

#%%viendo etiqueta buena

datos_test = np.hstack((datos_up,et_arauco))

#%%borrar hacia el lado
```

```
datos_as = np.asarray(datos_up)
datos_as_test = np.asarray(datos_test)

nuevos_datos = []
nuevos_datos_test = []
for row in range(datos_as.shape[0]):
    local_row = datos_as[row,:]
    local_row_test = datos_as_test[row,:]
    if (True in np.isnan(local_row)) == False:
    #and local_row[0] < 10**5: # index 2 is VAL356CI8017
        nuevos_datos.append(local_row)
        nuevos_datos_test.append(local_row_test)

nuevos_datos = np.asarray(nuevos_datos)
nuevos_datos_test = np.asarray(nuevos_datos_test)

#%%etiquetas de arauco

et_arauco_post_preprocesamiento = nuevos_datos_test[:,70]
et_post = []

for j in et_arauco_post_preprocesamiento:
    if j != 10:
        et_post.append(j)

et_post = pd.DataFrame(et_post)
et_buena = et_post.notnull()*1
et_buena = np.array(et_buena)
et_post = np.array(et_post)

for i in range(len(et_post)):
    if et_post[i] == 0:
        et_buena[i] = 0

et_buena = et_buena.astype(int) #etiquetas de arauco

#%%PCA

# Escalado para pca (feature_range = (0,1))
sc = StandardScaler()
dataset = sc.fit_transform(nuevos_datos)


#pca = PCA(n_components = dataset.shape[1])
pca = PCA(n_components = 14)
dataset_pca = pca.fit_transform(dataset)
```

```
variance_pca = pca.explained_variance_ratio_ # percentual variance
print(variance_pca)
print('suma:',sum(variance_pca))

#%%DB-SCAN

dbscan = DBSCAN(eps=4.1, min_samples=7000).fit(dataset_pca)

core_samples_mask = np.zeros_like(dbscan.labels_, dtype=bool)
core_samples_mask[dbscan.core_sample_indices_] = True


labels = dbscan.labels_
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

#%%preprando comparacion matriz de confusion

labels_fix = []
for i in labels:
    if i<0:
        labels_fix.append(i*-1)
    else:
        labels_fix.append(i)

labels_fix = np.asarray(labels_fix)
for j in range(len(labels_fix)):
    if labels_fix[j]>1:
        labels_fix[j]=1



et_predict = labels_fix[32045 : ]

#%%Definiendo matriz de confusi n

import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
```

```
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
    range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
#%%
 from sklearn.metrics import confusion_matrix
 conf_matrix = confusion_matrix(et_buena, et_predict)
 print('Confusion Matrix:\n',conf_matrix)

 plt.figure()
 plot_confusion_matrix(conf_matrix, classes=['Healthy','Anomaly'],
 normalize=True, title='Etiquetas ARAUCO')

#%%etiquetas

 in_cluster = []
 label_in_cluster = [] # 0: normal data
 out_cluster = []
 label_out_cluster = [] # 1: anomaly data

 for data_row in range(dataset.shape[0]):
 #and (dataset_pre[data_row,0]<=110)
    if labels[data_row] == 0: #and (dataset_pre[data_row,1]<=95)
    and (dataset_pre[data_row,2]<=12) and
    (dataset_pre[data_row,4]<=25):
        in_cluster.append(nuevos_datos[data_row,:])
        label_in_cluster.append(0)
```

```
        else:
            out_cluster.append(nuevos_datos[data_row,:])
            label_out_cluster.append(1)



 dataset_pcaf=pd.DataFrame(dataset_pca)
 columnas = list(dataset_pcaf)

 # In cluster Dataframe Info
 in_cluster_dataframe = pd.DataFrame(in_cluster, columns=columnas)
 in_cluster_describe = in_cluster_dataframe.describe()

 # Out cluster Dataframe Info
 out_cluster_dataframe = pd.DataFrame(out_cluster, columns=columnas)
 out_cluster_describe = out_cluster_dataframe.describe()

#%%Dataset with labels
 in_cluster = np.asarray(in_cluster)
 out_cluster = np.asarray(out_cluster)

 X_data = np.vstack((in_cluster, out_cluster))
 Y_data = np.vstack((np.zeros((in_cluster.shape[0],1)),
 np.ones((out_cluster.shape[0],1))))



#%%Training and Testing sets
 from sklearn.preprocessing import MinMaxScaler
 from sklearn.model_selection import train_test_split

 X_train, X_test, Y_train, Y_test = train_test_split(X_data,
 Y_data, test_size = 0.2)#, shuffle = False) #que no revuelva

 #scaler = StandardScaler()

 scaler = MinMaxScaler()
 X_train = scaler.fit_transform(X_train)
 X_test = scaler.transform(X_test)

#%%Guardando
 np.savez('Clasificacion.npz',
          x_train = X_train,
          x_test = X_test,
          y_train= Y_train,
          y_test = Y_test)
```

9.3. Código SVM

```
#%%SVM
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
#from sklearn.grid_search import GridSearchCV

Y_vec = Y_train.ravel() #dejar como vector sin []

svm_clf = Pipeline((("scaler", StandardScaler()),
("linear_svc", LinearSVC(C=1, loss = "hinge",
max_iter = 1000000))))#, verbose = True))))
svm_clf.fit(X_train,Y_vec)
svm_pred= svm_clf.predict(X_test)



#svm_clf.n_support_
#print (svm_clf.suppot_vectors_)
#parameters = {"SVM_C":[0.001, 0.1, 10, 100, 10e5],
"SVM_gamma":[0.1, 0.01]}

#grid = GridSearchCV(svm_clf, param_grid = parameters)



#%%Matriz de confusion


from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(Y_test, svm_pred)
print('Confusion Matrix:\n',conf_matrix)

plt.figure()
plot_confusion_matrix(conf_matrix, classes=['Healthy','Anomaly'],
normalize=True, title='Reboiler ARAUCO, SVM')
```

9.4. Código Modelo MLP

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
from keras.layers import Flatten
from keras.utils import np_utils
from keras.layers import SimpleRNN
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import TimeDistributed
from keras import optimizers


dataset = np.load('Clasificacion.npz')

X_train = dataset['x_train']
X_test = dataset['x_test']
y_train = dataset['y_train']
y_test = dataset['y_test']


#MLP

model = Sequential()

model.add(Dense(units = 20000, activation = 'relu', input_dim = 14))
model.add(Dropout(rate = 0.2))

model.add(Dense(units = 2000, activation = 'relu'))
model.add(Dropout(rate = 0.2))


model.add(Dense(units = 1000, activation = 'relu'))
model.add(Dropout(rate = 0.2))


model.add(Dense(units = 500, activation = 'relu'))
model.add(Dropout(rate = 0.2))
```

```
model.add(Dense(units = 250, activation = 'relu'))
model.add(Dropout(rate = 0.2))


model.add(Dense(units = 125, activation = 'relu'))
model.add(Dropout(rate = 0.2))


model.add(Dense(units = 75, activation = 'relu'))
model.add(Dropout(rate = 0.2))


model.add(Dense(units = 30, activation = 'relu'))
model.add(Dropout(rate = 0.2))


model.add(Dense(units = 15, activation = 'relu'))
model.add(Dropout(rate = 0.2))


model.add(Dense(units = 7, activation = 'relu'))
model.add(Dropout(rate = 0.2))

#capa salida
model.add(Dense(units = 1, activation = 'sigmoid'))

model.compile(optimizer = 'adam', loss = 'binary_crossentropy',
metrics = ['accuracy'])
model_history = model.fit(X_train, y_train, batch_size = 785,
epochs = 200, validation_split = 0.3)#, shuffle = True)

#%%#########################################################
#### Prediciendo el modelo y evaluandolo ##########
#########################################################


y_pred = model.predict_classes(X_test)


#%%Evaluando el modelo

test_loss, test_accuracy = model.evaluate(X_test, y_test,
batch_size = 785)


#%%Definiendo matriz de confusion
```

```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
    range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

#%%Matriz de confusi n


from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n',conf_matrix)

plt.figure()
plot_confusion_matrix(conf_matrix, classes=['Healthy','Anomaly'],
normalize=True,
                      title='Reboiler ARAUCO')
```

```
#%%Gr fico de perdida training y validation set en
 funci n de las epoch

 plt.plot(model_history.history['loss'])
 plt.plot(model_history.history['val_loss'])
 plt.title('MLP model loss')
 plt.ylabel('Loss')
 plt.xlabel('Epoch')
 plt.legend(['train', 'validation'], loc='upper right')
 plt.show()

#%%Grafico 4 cosas


 plt.style.use("ggplot")
 plt.figure()
 N = 200
 plt.plot(np.arange(0, N), model_history.history["loss"],
 label="train_loss")
 plt.plot(np.arange(0, N), model_history.history["val_loss"],
 label="val_loss")
 plt.plot(np.arange(0, N), model_history.history["acc"],
 label="train_acc")
 plt.plot(np.arange(0, N), model_history.history["val_acc"],
 label="val_acc")
 plt.title("Training Loss and Accuracy")
 plt.xlabel("Epoch #")
 plt.ylabel("Loss/Accuracy")
 plt.legend(loc="lower left")
 plt.show()
```

9.5. Código Modelo CNN

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

```
from keras.layers import Flatten
from keras.utils import np_utils
from keras.layers import SimpleRNN
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import TimeDistributed
from keras import optimizers




dataset = np.load('Clasificacion.npz')
X_train_pre_img = dataset['x_train']
X_test_pre_img = dataset['x_test']
X_train = np.reshape(X_train_pre_img,[-1,1,14,1])
#Se cambia el tama o de la 'imagen' que queremos (el 1 del final).
X_test = np.reshape(X_test_pre_img,[-1,1,14,1])
y_train = dataset['y_train']
y_test = dataset['y_test']

#%%CNN


model = Sequential()


# capa entrada + primera capa
model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', input_shape = (1,14,1), padding = 'same'))
#model.add(MaxPooling2D(pool_size=(5,5)))
model.add(Dropout(rate = 0.2))


model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', padding = 'same'))
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(rate = 0.2))


model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', padding = 'same'))
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(rate = 0.2))

model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', padding = 'same'))
```

```
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(rate = 0.2))

model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', padding = 'same'))
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(rate = 0.2))


model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', padding = 'same'))
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(rate = 0.2))


model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', padding = 'same'))
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(rate = 0.2))

model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', padding = 'same'))
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(rate = 0.2))))

model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', padding = 'same'))
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(rate = 0.2))

model.add(Conv2D(filters = 256, kernel_size = (1,7), activation =
'relu', padding = 'same'))
#model.add(MaxPooling2D(pool_size=(1, 2)))
model.add(Dropout(rate = 0.2))



model.add(Flatten()) #se usa si la ultima capa usamos
return_sequences = True, se omite con false
(se pone como comentario)

# MLP post flatten
model.add(Dense(units = 500, activation = 'relu'))
model.add(Dense(units = 250, activation = 'relu'))
model.add(Dense(units = 175, activation = 'relu'))
```

```
# capa salida
#adam = optimizers.Adam(lr = 0.00001, beta_1 = 0.9, beta_2 =
0.999, epsilon = None, decay = 0.0, amsgrad = False)
#sgd = optimizers.SGD(lr = 0.000000000001, decay = 1e-6, momentum
= 1.9)
model.add(Dense(units = 1, activation = 'sigmoid'))



model.compile(optimizer = 'adam', loss = 'binary_crossentropy',
metrics = ['accuracy'])


model_history = model.fit(X_train, y_train, batch_size = 785,
epochs = 200, validation_split = 0.3)#, shuffle = True)

#%%#########################################################
#### Prediciendo el modelo y evaluandolo ##########
#########################################################


y_pred = model.predict_classes(X_test)


#%%Evaluando el modelo

test_loss, test_accuracy = model.evaluate(X_test, y_test,
batch_size = 785)


#%%Definiendo matriz de confusion
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)
```

```
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
    range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()

#%%Matriz de confusion


from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print('Confusion Matrix:\n',conf_matrix)

plt.figure()
plot_confusion_matrix(conf_matrix, classes=['Healthy','Anomaly'],
normalize=True, title='Reboiler ARAUCO')


#%%Grafico de perdida training y validation set en funcion de las epoch

plt.plot(model_history.history['loss'])
plt.plot(model_history.history['val_loss'])
plt.title('CNN model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()



#%%Grafico accuracy
```

```
plt.plot(model_history.history['accuracy'])

plt.plot(model_history.history['val_accuracy'])
plt.title('CNN model accuracy')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'validation'], loc='lower right')
plt.show()




#%%Grafico 4 cosas


plt.style.use("ggplot")
plt.figure()
N = 200
plt.plot(np.arange(0, N), model_history.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), model_history.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), model_history.history["acc"], label="train_acc")
plt.plot(np.arange(0, N), model_history.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.show()

#plt.savefig(args["plot"])
```

9.6. Pantalla de operación sistema bombeo

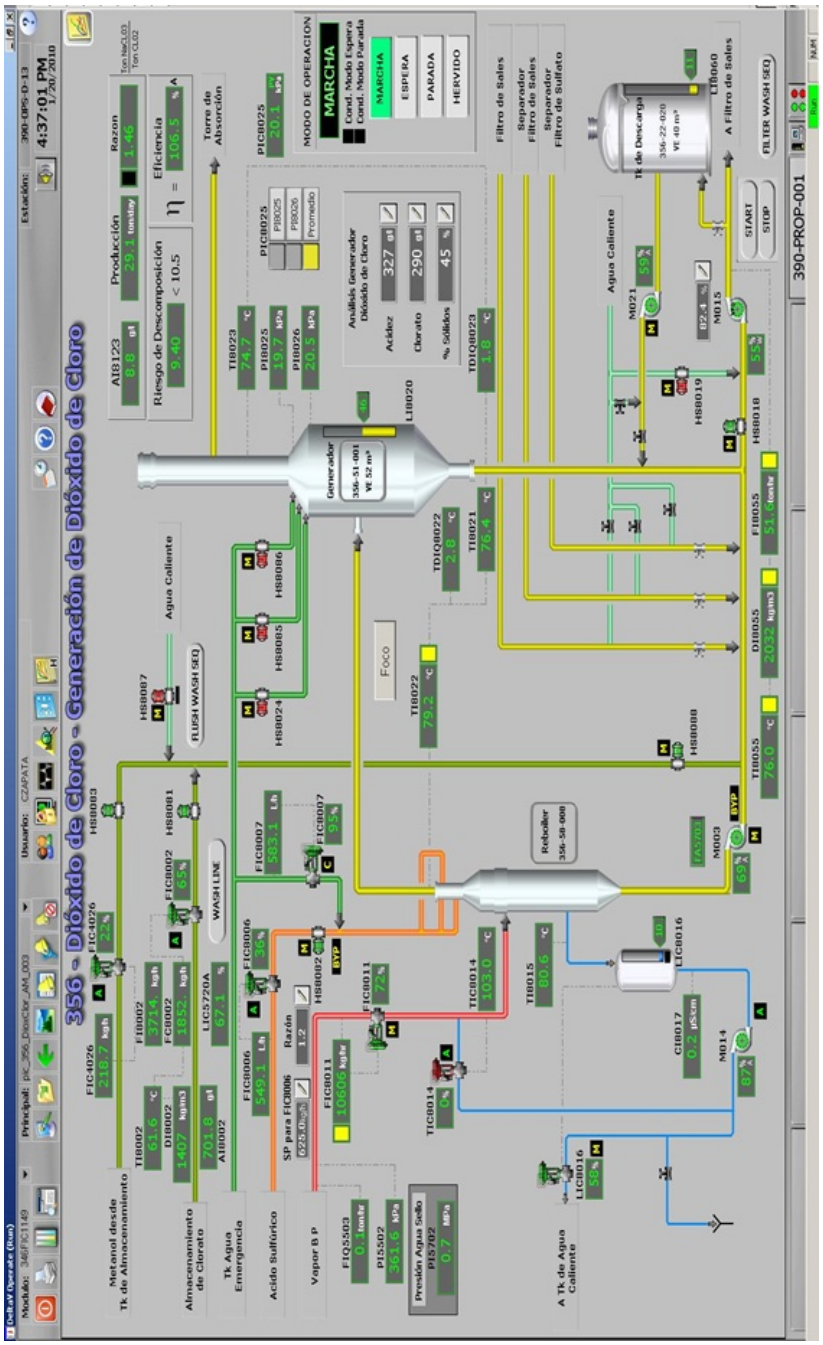

Figura 9.1: Pantalla operación [19]