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

REDES NEURONALES CONVOLUCIONALES BAYESIANAS PARA DIAGNOSTICOS DE FALLA EN ACTIVOS FÍSICOS BAJO INCERTIDUMBRE: CASO DE ANALISIS EN BOMBAS CENTRIFUGAS.

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO

MATÍAS ANDRÉ MARÍN MARCHANT

PROFESOR GUÍA
ENRIQUE LOPEZ DROGUETT

MIEMBROS DE LA COMISIÓN
JUAN TAPIA FARIAS
VIVIANA MERUANE NARANJO

SANTIAGO DE CHILE

2020

RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO

POR: MATÍAS ANDRÉ MARÍN MARCHANT

FECHA: 2020

PROF. GUÍA: ENRIQUE LOPEZ DROGUETT

REDES NEURONALES CONVOLUCIONALES BAYESIANAS PARA DIAGNOSTICOS DE FALLA EN ACTIVOS FÍSICOS BAJO INCERTIDUMBRE: CASO DE ANALISIS EN BOMBAS CENTRIFUGAS.

El campo del Machine Learning avanza rápidamente en pos de mejorar el desarrollo de inteligencias artificiales con múltiples fines. Para esto, desarrollar aprendizaje en Deep Learning y Redes Neuronales resulta fundamental para poder analizar volúmenes de datos más grandes y complejos. Para cumplir este objetivo, se utilizan un gran número de técnicas de desarrollo de algoritmos computacionales para así poder programar redes capaces de aprender iterativamente sobre los datos. Pero esta metodología no está falta de problemas y estos problemas ponen en jaque la confiabilidad de las redes neuronales diseñadas. Para solucionar esto, se implementa un nuevo tipo de red que, basándose en conceptos de incertidumbre (Bayes), permitiría aumentar la confiabilidad y utilidad de los resultados obtenidos permitiendo una mejor respuesta a los datos proporcionados.

En el presente trabajo se muestran los resultados de trabajar con redes neuronales sobre una base de datos, con el objetivo de realizar diagnósticos de falla agregando incertidumbre al sistema, para poder analizar escenarios distintos al que los datos plantean. Mediante el uso de un detector de anomalías y un clasificador, 2 algoritmos tradicionales utilizados en redes neuronales, se obtienen resultados iniciales para los datos

Dedicado a aquellos que eligieron compartir sus vidas conmigo, hacen mi vida mucho más entretenida, y siempre estaré agradecido con ustedes.

Para aquellos que lean esto en el futuro, esta tesis fue hecha en uno de los periodos más difíciles del país, con revoluciones sociales y una pandemia global. Disfruten.

La universidad es difícil, y ha sido un trayecto largo para llegar hasta aquí, así que los agradecimientos son largos.

A mi familia, por estar siempre ahí, creyendo que puedo ser mejor que lo que soy y que puedo lograr las metas, que por más que no tenga las ganas o no me parezcan divertidas, las tareas hay que hacerlas y estar orgulloso del resultado.

A Catalina, por ser un rayo de luz en mi vida, yo sé que al momento que escribo esto, no estás pasando por un buen momento, pero créeme que, sin ti yo estaría peor. Soy una mejor persona gracias a ti, y siempre te estaré agradecido por ello.

Al Anime no Seishin Doukoukai, cuando llegué a Beauchef, estaba destinado a tener una vida donde no podría apreciar las opiniones del resto debido a que tenía un ego inflado. Aquí aprendí la importancia de las opiniones del resto y de cómo esto me ayudaría a ser una mejor persona y profesional. Seishin es un lugar sagrado que nunca se puede perder, ya que permite que todos los que disfrutan de la cultura japonesa se sientan acogidos y tengan un espacio para compartir con otros que tienen los mismos gustos.

Al profesor Enrique, por darme esta oportunidad y presentarme este mundo loco del Deep Learning, de verdad es interesante aprender un tópico que evoluciona al mismo tiempo que uno escribe esto, es definitivamente un desafío que nunca olvidaré.

Al equipo de la Sala de Posgrado, tienen mi admiración eterna por el nivel de comprensión que poseen de un tópico tan difícil y cambiante como el Deep Learning.

Tabla de Contenido

Capítulo 1: Introducción

El Deep Learning es una sub-clase del Machine Learning que permite enseñar procedimientos a máquinas mediante el uso de algoritmos computacionales. En la actualidad, este concepto se utiliza para que máquinas puedan aprender a diagnosticar procesos y estados de falla de manera independiente utilizando los conceptos de redes neuronales que se basan en replicar los procesos neuronales humanos para su implementación en máquinas.

A medida que la complejidad de los problemas crece, las necesidades de aprendizaje de las máquinas también y para lograr esto, se mejoran las redes neuronales disponibles. El resultado de esto fueron las redes neuronales convolucionales. Ahora, con la necesidad de que las máquinas aprendan a manejar incertidumbre, las redes neuronales bayesianas son la nueva alternativa de la ciencia, redes capaces de analizar intervalos de probabilidad y así las máquinas son capaces de tomar decisiones basadas en intervalos de incertidumbre.

El presente informe incluye la investigación realizada con el objetivo general de poder presentar la posibilidad de usar algoritmos de redes neuronales convolucionales bayesianas que puedan asistir máquinas a establecer sus propios regímenes de mantenimiento. Dentro de este objetivo general, existen objetivos específicos donde el primero es crear un detector de anomalías y un algoritmo de diagnóstico de fallas para analizar los datos. Para posteriormente crear versiones de estos que incluyan incertidumbre para ver como la incertidumbre afecta los resultados.

El estudio se hace en bombas centrifugas, debido a que son altamente utilizadas en la industria petrolera, tanto en los trabajos de extracción de crudo como en el transporte de materia prima o en el manejo de los fluidos utilizados para los procesos de transformación del petróleo. Permitiendo que, basándose en datos de años anteriores, éstas puedan realizar análisis automáticos de estado y diagnosticar fallas, además de poder generar sus propios procedimientos de mantenimiento gracias al manejo de incertidumbre mediante el uso de redes neuronales bayesianas.

Capítulo 2: Marco Teórico

Deep Learning

Deep Learning es una metodología que permite a las computadoras aprender mediante ejemplos, imitando a los humanos. La gran ventaja de lograr esto es la cantidad de datos que las máquinas pueden procesar, además de que procesan los datos de manera más rápida y precisa que los humanos jamás podrían procesarlos. [1]

Actualmente esta tecnología se está utilizando para múltiples fines, ejemplos notorios son los autos inteligentes que se manejan solos [2], investigaciones médicas en pos de la detección de células cancerosas [1].

Existen 2 tipos de Machine Learning:

Aprendizaje Supervisado

El aprendizaje supervisado ocurre cuando el conjunto de datos entregado al algoritmo diseñado es proporcionado por el laboratorio realizando la investigación. Usualmente estos datos consisten en información obtenida a lo largo de múltiples intervalos de tiempo y donde la relación entrada-salida es conocida, es decir, que se conocen los resultados de ingresar estos datos, por lo que el objetivo de este método es que el algoritmo sólo tiene que replicar la ruta al resultado y aprenderla.

Tipos de problemas donde se usa aprendizaje supervisado:

- Clasificaciones: Donde los algoritmos entregan un resultado de estado (Negro/Blanco, Verdadero/Falso).
- Regresiones: Donde los algoritmos entregan variables en unidades de medida conocidas (Kg, CLP, m).

Ejemplos de aprendizaje supervisado:

- Máquinas de Vectores de Soporte (SVM): Son un modelo de aprendizaje supervisado de clasificación de datos, que consiste en una vez entregados datos al modelo, se generan 2 clasificaciones, lo que lo convierte en un clasificador no-probabilístico binario.

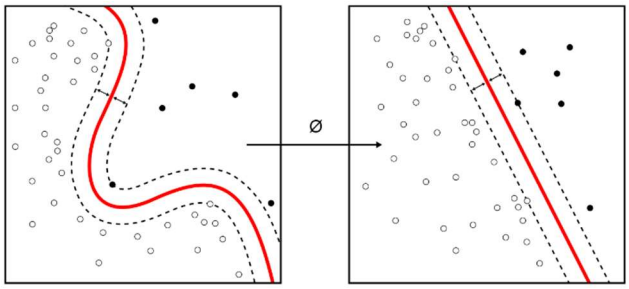

Fig 1: Representación gráfica del funcionamiento de una SVM.

- Bosque de decisiones aleatorio: El método del bosque de decisiones aleatorio consiste en generar una serie de árboles de decisiones con todos los valores posibles de manera de poder predecir mediante resultados aleatorios del bosque.

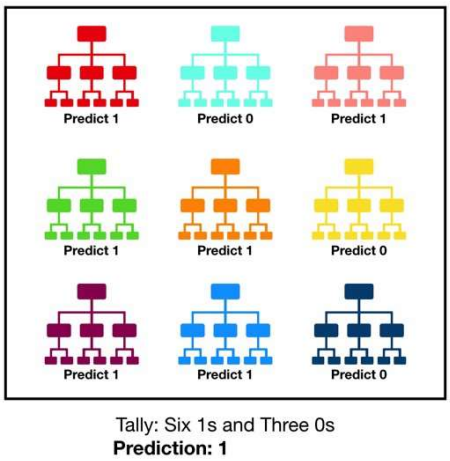

Fig 2. Descripción gráfica de un Bosque Aleatorio. Fuente: Towards Data Science.

Aprendizaje No Supervisado

El aprendizaje no supervisado ocurre cuando sólo se tienen los datos de entrada y los datos de salida no existen. Este tipo de aprendizaje se utiliza con el objetivo que el algoritmo modele la estructura interna de los datos y así aprender más sobre éstos.

Tipos de problemas donde se usa aprendizaje no supervisado:

- Aglomeración: Donde el objetivo es descubrir posibles grupos dentro de los datos y clasificarlos.

- Asociación: Donde el objetivo es encontrar reglas que describan la mayor porción posible de los datos.

Ejemplos de aprendizaje no supervisado

- Autoencoders: Los autoencoders son un tipo de aprendizaje no supervisado donde la red aprende mediante la reproducción de datos entregados, donde primero deconstruye los datos mediante un codificador (encoder) y posteriormente los reconstruye mediante un decodificador (decoder).

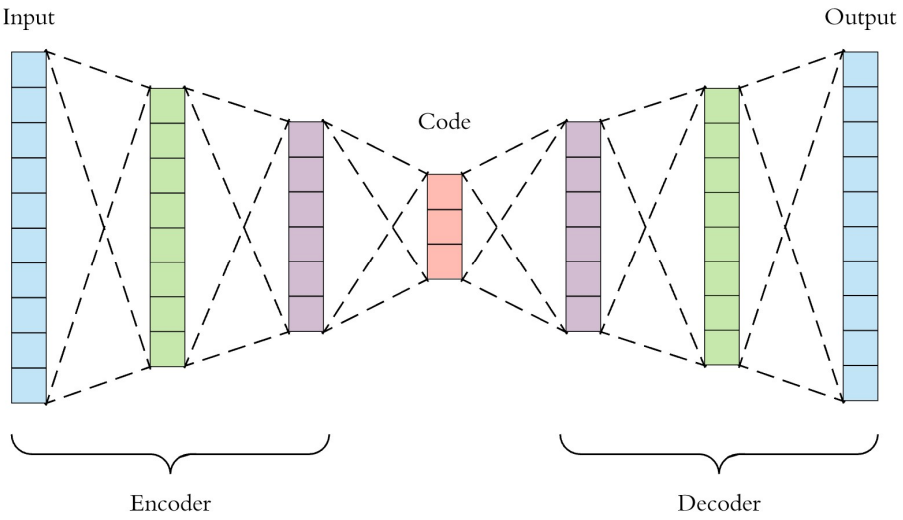

Fig 3. Representación genérica de un Autoencoder. Fuente: Towards Data Science.

Redes Neuronales Artificiales

El objetivo principal del Deep Learning es duplicar los procesos de sinapsis del cerebro humano (fig. 1), esto se logra mediante la creación de redes neuronales artificiales que puedan aprender mediante ejemplos y con la menor cantidad de reglas limitantes.

Los componentes de una red neuronal son los siguientes:

- Neuronas: Son los componentes que reciben los datos y los transmiten a otras neuronas, las neuronas que inician el proceso de comunicación se llaman neuronas de entrada. A su vez, las neuronas tienen sus propios componentes:
  - Activación $a_j(t)$: El estado actual de la neurona dependiendo de una variable de tiempo.
  - Un umbral $\Theta(t)$: Se mantiene fijo a menos que sea influenciado por otros componentes.
  - Una función de activación f que computa la nueva activación para t+1.
  - Y una función de salida f_{out} que computa la salida obtenida de la activación. $o_j(t)= f_{out}(a_j(t))$
- Conexiones: Son los componentes que conectan cada neurona.
- Pesos: Son valorizaciones aplicadas a los componentes de cada neurona de manera de representar la importancia de cada parte.
- Sesgos: Son valorizaciones aplicadas posteriormente a la aplicación de todos los pesos para generar los umbrales para la función de activación.
- Función de propagación: Esta función computa la entrada a la siguiente neurona basándose en la salida generada por la neurona anterior. En este punto se pueden agregar sesgos.

  $p_j(t) = \sum_i o_i(t)\, w_{ij} + w_{0j}$ donde w_{0j} es el sesgo utilizado.
- Regla de aprendizaje: Algoritmos que modifican los parámetros de la red favoreciendo cierto tipo de resultados.

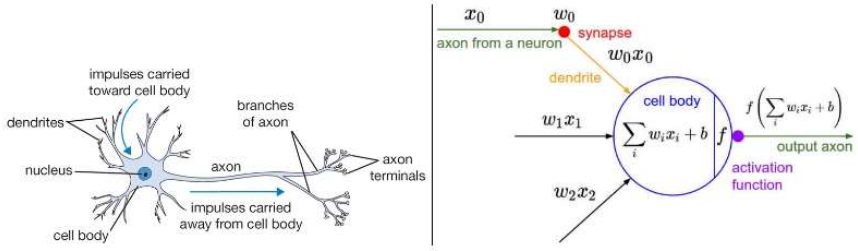

Fig 4: Comparación entre la sinapsis cerebral y una red neuronal artificial. Fuente: [3].

Redes Neuronales Convolucionales

Una red neuronal convolucional es un algoritmo de redes neuronales capaz de procesar imágenes, asignar importancias a objetos de la imagen y diferenciarlos de otros objetos mediante un núcleo o kernel de procesamiento que permite realizar los cálculos necesarios.

Una de las características principales de las redes neuronales convolucionales es que las neuronas están distribuidas en arreglos tridimensionales y en capas que se superponen una sobre otra. Los componentes principales son los siguientes:

- Una capa de entrada: La imagen inicial a ser analizada.
- Una capa de salida: El arreglo de resultados obtenido.
- Capas ocultas intermedias: Existen múltiples tipos de capas ocultas:
    - Capas convolucionales: Estas capas se dedican a capturar las partes de alto nivel de la imagen, pueden ser múltiples capas convolucionales y cada una remueve características como bordes, colores o gradientes.
    - Capas rectificadoras lineales (RELU): Es la función de activación de la neurona, se representa como:
      $f(x) = x^+ = Max(0,x)$
    - Capas de agrupación: Estas se dedican a disminuir el espacio ocupado por la figura ingresada, con el objetivo de reducir el poder computacional para procesar los datos y de disminuir ruido [4].
    - Capas conectadas completamente: Estas capas tienen todas las neuronas conectadas entre sí, permiten la clasificación de las imágenes.

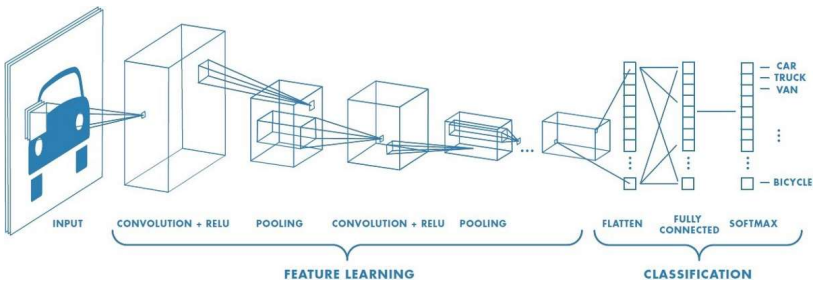

Fig. 5: Diagrama básico de una red neuronal convolucional. Fuente [4]

Redes Neuronales Bayesianas

Las redes neuronales bayesianas (BNN) salen de la necesidad de prevenir el fenómeno de sobreajuste, combinando la utilidad de las redes neuronales y de los modelos estocásticos. Usar BNN involucra introducir incertidumbre al proceso de aprendizaje del algoritmo con el objetivo de que éste pueda alcanzar garantías probabilísticas en las predicciones [5].

La idea principal de las BNN está basada en el teorema de Bayes para probabilidades.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Fig. 6: Definición del teorema de Bayes.

Este teorema se utiliza de manera de poder dar solución a problemas más complejos utilizando conocimientos previos y generando pesos distintos en cada una de las capas ocultas en forma de distribuciones de probabilidades y así usarla como entrada a nuevas capas ocultas que mejoran el refinamiento del algoritmo. El resultado final de este proceso es una distribución de probabilidades con una mayor precisión predictiva y nivel de confianza.

Existen 2 tipos de incertidumbre comúnmente utilizados:

- Aleatoria: Mide el impacto del ruido inherente sobre el análisis, este tipo de incertidumbre no puede ser reducida independiente de la cantidad de muestras. Se puede modelar artificialmente utilizando distribuciones en la salida del algoritmo.
- Epistémica: Es la incertidumbre generada por el modelo en sí, es posible reducirla mediante más iteraciones del modelo. Se puede modelar artificialmente agregando distribuciones inmediatamente después de la entrada del algoritmo.

La metodología sobre cómo se aplica el teorema de Bayes a las redes neuronales, consiste en convertir los pesos y sesgos de cada red neuronal, en distribuciones aleatorias. Es decir, que el valor de cada peso y sesgo esté determinado por una variable aleatoria que va a ser distinta cada vez que el algoritmo se ejecute. Esto es representado en la siguiente figura, donde se muestra como los resultados de una red desarrollada cambian a distribuciones debido a la influencia de Bayes.

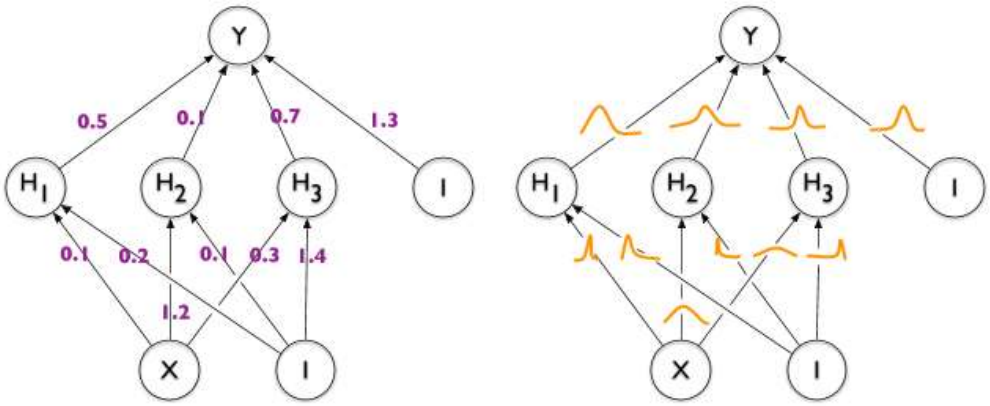

Fig 7: Figura explicativa del impacto de trabajar con influencia bayesiana.

Muestreo

El proceso de obtener nuevos resultados desde inputs aleatorios se conoce como muestreo. Donde cada una de las posibles entradas generadas por la distribución de probabilidades es cuantificada, lo que genera como resultado una distribución de probabilidades para los valores de salida, lo que permite establecer intervalos de confianza e incertidumbre para cada valor de salida del modelo.

El objetivo final del muestreo es que después de una gran cantidad de iteraciones, es posible detectar que la incertidumbre obtenida de cada iteración es muy alta, lo que es una forma que tiene la red de decir: “no sé de qué se trata esta foto”. [6]

Muestreo por importancia

El objetivo de esta metodología es poder optimizar el aprendizaje en procesos donde el tiempo es limitado. Dentro del proceso de muestreo, ocurre que no todas las muestras tienen igual relevancia, por lo que se pueden establecer métodos que permiten discriminar parte de las muestras. El paso más importante es poder comprobar que al utilizar este método no se pierde la eficacia del entrenamiento de la red neuronal.

Sobreajuste y subajuste

Sobreajuste es un fenómeno que ocurre a medida que se van generando más iteraciones del algoritmo, donde el modelo se acostumbra tanto a los datos de entrada que la precisión de los resultados deja de ser la esperada ya que el objetivo del modelo es que interactúe de manera precisa para cualquier conjunto de datos. Por otro lado, el subajuste es cuando los resultados del

modelo no mejoran a pesar de tener capacidad de mejorar, causas de esto típicamente se asocia a que el modelo no ha sido entrenado lo suficiente o no es lo suficientemente poderoso para completar la tarea.

Para combatir el sobreajuste lo mejor es tener más datos de entrenamiento, mientras más datos se tengan, mejor aprenderá el modelo. Otra solución viable es reducir el tamaño del modelo, es decir la cantidad de parámetros que el modelo deba aprender, o tener que aprender mediante representaciones más compactas.

Bombas Centrifugas

Las bombas centrifugas son máquinas utilizadas para transportar fluidos transformando energía cinética en energía hidrodinámica entregada por el movimiento del fluido. En la industria estas bombas son escogidas por su ingeniería simple, capacidad de trabajar con soluciones abrasivas y la capacidad de transportar altas cantidades de fluidos. Y es su uso constante en la industria la razón de la importancia de esta investigación.

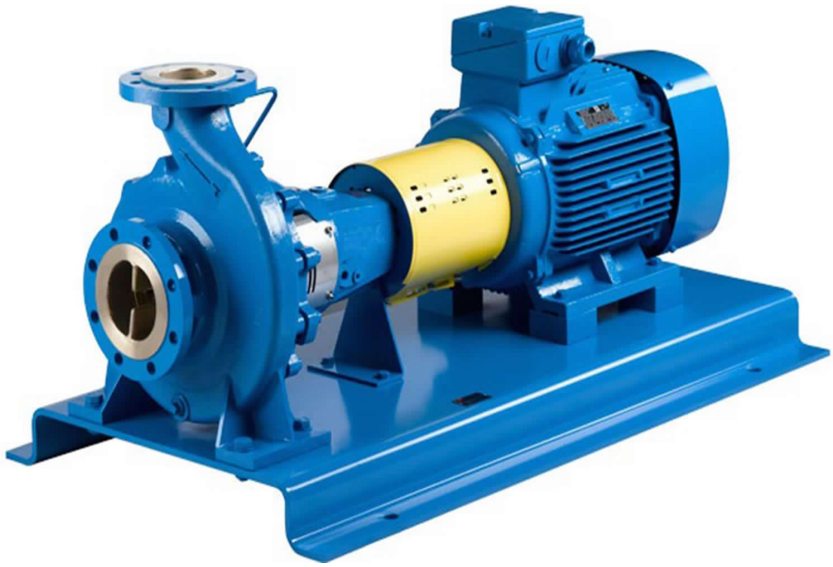

Fig. 8: Bomba centrifuga genérica. Fuente: [7]

Parámetros principales

Para trabajar con bombas centrifugas, los parámetros a controlar durante el presente trabajo de título son:

- Desalineamiento vertical.
- Desalineamiento horizontal.
- Desbalance.
- Numero de fallas del circuito externo en rodamientos.
- Numero de fallas de la jaula en los rodamientos.
- Numero de fallas de las bolas en los rodamientos.

Capítulo 3: Descripción de la problemática

La problemática abarca una bomba centrifuga de motor eléctrico perteneciente a la empresa Petrobras, utilizada en una de sus plantas de extracción de petróleo, opera con agua de mar, y su funcionamiento se muestra en la siguiente figura:

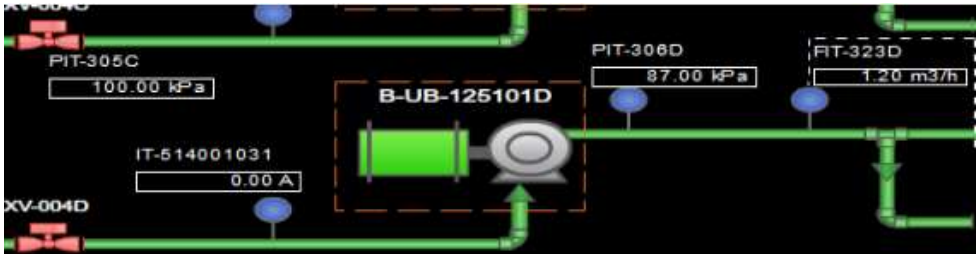

Fig. 9. Diagrama de operación de la bomba B-UB-125101D. Fuente: Petrobras.

Esta bomba es controlada por sensores ubicados en distintos lugares de esta, donde se controlan los siguientes factores:

- Desalineamiento horizontal.
- Desbalance.
- Numero de fallas del circuito externo en rodamientos.
- Numero de fallas de la jaula en los rodamientos.
- Numero de fallas de las bolas en los rodamientos.

Todos estos datos vienen incluidos dentro de una base de datos que controla los 17 sensores de la bomba, el diagrama de estos sensores se muestra en la siguiente figura:

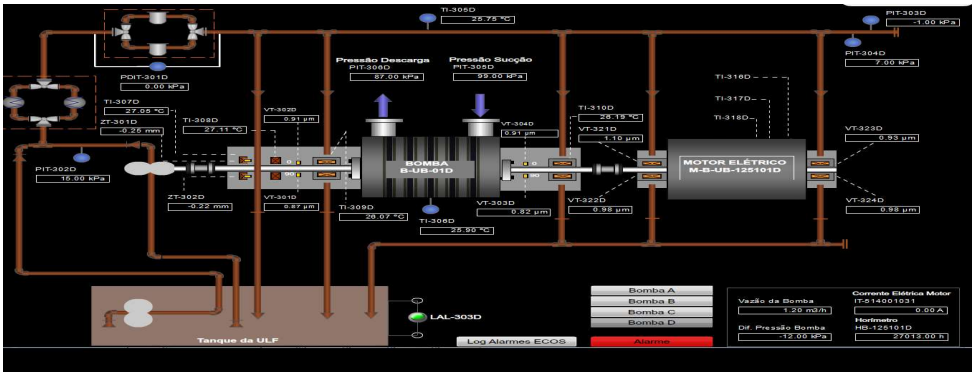

Fig. 10: Diagrama de los sensores de la Bomba B-UB-125101D de Petrobras.

Dentro de los datos, las matrices X (x_train, x_test) contienen los estados de cada uno de los 17 sensores a lo largo de los intervalos temporales distribuidos durante 3 años de servicio. Donde se presentan con los siguientes valores:

0 = Sensor no está activo.

1 = Sensor está activo.

Junto con los datos de los sensores se cuenta con arreglos de datos Y (que contienen y_train, y_test) que se ve de la siguiente forma:

a	1	2	3	4	5	6	...
b	0	1	0	3	1	2	...

Fig. 11: Representación de los datos Y.

Donde la primera fila de datos corresponde a los intervalos temporales de la medición y la segunda fila corresponde a una clase que representa el estado de la maquina en el instante donde se sigue la siguiente definición basada en la Norma ISO 14224 “Recolección de Petróleo y Gas Natural – Recolección e Intercambio de datos de confiabilidad y mantenimiento de equipos”:

0 = Estado normal: La bomba está en un estado normal, funcionando sin fallas.

1 = Estado Dañado: La bomba no está funcionando adecuadamente, hay fallas detectadas en algunos de los sensores.

2 = Estado Crítico: La bomba posee fallas graves detectadas en algunos de los sensores.

3 = Estado Catastrófico: Las fallas detectadas en los sensores pueden causar daño permanente a la bomba centrifuga si se mantienen.

Capítulo 4: Hipótesis

El presente trabajo plantea desarrollar e implementar un algoritmo de Bayesian Deep Learning para demostrar que agregar incertidumbre a redes neuronales permite un mejor análisis de los resultados obtenidos mediante este método en comparación a los obtenidos a través de una red neuronal tradicional.

Esta Hipótesis es respaldada por artículos que muestran que agregar incertidumbre a las redes permite la entrada a análisis más profundos y complejos sobre los mismos grupos de datos. Ya que la intención del Bayesian Deep Learning es generar intervalos de confianza que contengan el valor correcto [10].

Capitulo 5: Metodología

Establecer las librerías:

El primer paso del trabajo fue aprender a utilizar las librerías de Python adecuadas para realizar los procesos de Deep Learning. Para este trabajo se utilizaron las siguientes librerías:

- Tensorflow: Permite la construcción de redes neuronales.
- SciPy: Permite el uso de herramientas de cálculo básicas en Python.
- Scikit-learn: Permite el uso de algoritmos de clasificación, regresión y empaquetamiento.
- NumPy: Permite la manipulación de grandes vectores y matrices de datos para realizar cálculos.
- Matplotlib: Permite la construcción de gráficos y elementos visuales.
- Jupyter Notebook: Plataforma que permite la escritura de códigos de Python, es decir, la plataforma donde se escribieron los códigos utilizados.
- Pandas: Permite el análisis de datos mediante tablas de datos y series temporales.

De estas librerías, Tensorflow y Jupyter Notebook son las más importantes ya que Tensorflow otorga las herramientas para la construcción de las redes neuronales necesarias y Jupyter Notebook otorga el entorno donde poder escribir el código.

Además, dentro de Tensorflow, se utilizó la librería de Tensorflow Probability, que permite el cálculo de probabilidades dentro de las redes neuronales creadas mediante Tensorflow, lo que permite añadir los escenarios de incertidumbre utilizados en este trabajo.

Todos los códigos utilizados en este trabajo se encuentran en los anexos.

Establecer los Modelos

Luego de establecer las librerías a utilizar, el siguiente paso consistió en definir los modelos a utilizar para resolver la problemática. En este trabajo se procedió con el siguiente curso de acción:

Buscar una red neuronal que funcione para los datos => Agregar incertidumbre en ella.

Para esto se definió que cada uno de los 2 procesos definidos en los objetivos serían tratados en orden, con la detección de anomalías siendo la primera en ser tratada y con el diagnostico de fallas siendo tratado después.

Detección de anomalías

Para la detección de anomalías se estableció que el mejor modelo a utilizar es un Autoencoder, un tipo de red neuronal que replica datos entregados mediante un proceso de compresión y descompresión de datos.

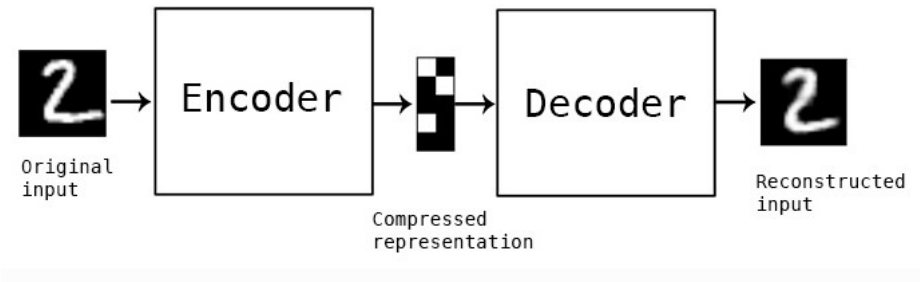

Fig. 12. Representación gráfica de un Autoencoder.

Los autoencoders son específicos a los datos entregados, y la respuesta que entregan es un intento de copia de los datos originales. La utilidad de la respuesta de un Autoencoder es lo que los hace útiles para detección de anomalías, la capacidad de aprender y reconstruir un cierto conjunto de datos permite detectar las anomalías presentes en datos que presentan ruido respecto del conjunto de datos iniciales.

Otro buen argumento para utilizar autoencoders, es que pueden analizar datos independientes del tamaño, clase y dimensiones, en comparación con otros métodos como SVM o Bosque Aleatorio que están limitados a objetos con clase binaria desde el principio. Lo que exigirá un tratamiento de datos que podría llevar a resultados no apropiados.

Para visualizar los resultados del uso del Autoencoder, se calculó un error cuadrático medio (mean squared error = mse) y posteriormente un error de reconstrucción, es decir, la diferencia entre el valor real y el reconstruido por el Autoencoder. Con ese error de reconstrucción, se construyeron gráficos que lo representan respecto a los intervalos temporales.

Diagnóstico de Fallas

Para el diagnostico de fallas, el método escogido incluyó un clasificador, un tipo de red neuronal que permite diferenciar las clases para una gran cantidad de objetos en un arreglo. Dado que uno de los grupos de arreglos presentes en los datos (Y) pose la forma adecuada para ser analizada mediante este método, se utilizó para clasificar los estados de los datos entregados.

Para visualizar los resultados, se obtuvo una matriz de confusión, que representa cómo la red ha clasificado los valores del arreglo, donde cada valor por encima de la diagonal es un falso positivo y cada valor por debajo de la diagonal es un falso negativo, como se ve en la figura anterior.

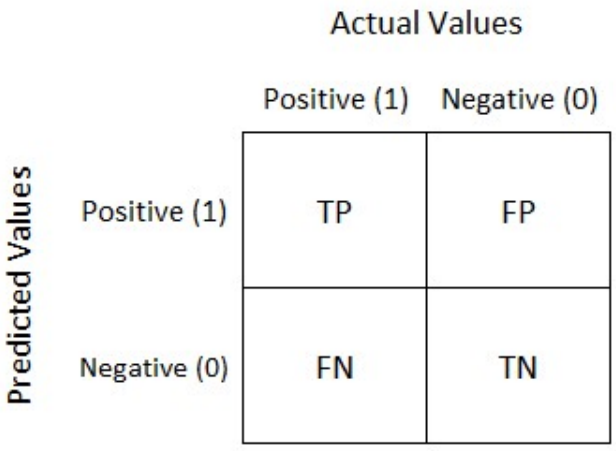

Fig. 13: Ejemplo de Matriz de Confusión.

Gracias a estos resultados, un clasificador se escogió como el método adecuado para el diagnóstico de fallas.

Preparación de los datos

Para asegurar que los métodos escogidos resultan adecuados, se probó un detector de anomalías y un clasificador con los datos utilizados en esta tesis. Los resultados obtenidos son archivados para poder realizar una comparación directa una vez añadida la incertidumbre a las redes.

Agregando incertidumbre a los modelos

Agregar incertidumbre a una red neuronal consiste en tomar la función objetivo optimizada por una red neuronal y agregarle un sesgo:

Siendo:

$$(Y - w * X)^2$$

la función objetivo de una regresión lineal, se convierte en:

$$p(Y \mid X, w) * p(w)$$

Posteriormente, esta se optimiza mediante muestreo y estimación, lo que matemáticamente se hace mediante una aproximación de Monte Carlo

$$E[\, p(x, w) / q(w) \,] \approx sample\ mean[\, p(x, w) / q(w) \,]$$

Y posteriormente, se calcula la inferencia variacional sobre:

$$q(w \mid v) \approx p(w \mid X, Y)$$

El siguiente paso es utilizar la divergencia de Kullback-Leibler y obtener la inferencia variacional.

$$KL[q \,||\, p] = E[\, log\ (q / p) \,]$$

Y así se descompone la data y la Evidencia del límite inferior (Evidence Lower Bound = ELBO) para finalmente utilizar la integración de Monte Carlo para optimizar el ELBO.

Este proceso matemático define la idea principal detras todas las distintas maneras de añadir incertidumbre a las redes neuronales, lo que será diferente dependiendo de cada algoritmo utilizado.

Para el detector de anomalías, se cambia el algoritmo utilizado de un Autoencoder convencional a un Autoencoder variacional, lo que añadió su propio grupo de desafíos ya que estos usan una arquitectura distinta a la de uno convencional, y tuvo que ser adecuada al set de datos utilizado. El uso del Autoencoder variacional, genera la incertidumbre mediante una distribución gaussiana agregando la interferencia a los pesos de la red.

Para el clasificador se agrega una capa llamada DenseFlipout, que funciona de la misma forma que una capa Dense, pero con un ruido agregado, lo que agrega la incertidumbre necesaria para el experimento.

Capítulo 6: Resultados y Análisis

Detección de Anomalías

Para la detección de anomalías se utilizaron los siguientes hiperparámetros:

- Numero de Capas: 7.
- Tamaño de batch: 256.
- Número de épocas: 100.

Utilizando una red con la siguiente arquitectura, se ejecuta el programa:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 24, 16)            288
_________________________________________________________________
dropout (Dropout)            (None, 24, 16)            0
_________________________________________________________________
dense_1 (Dense)              (None, 24, 3)             51
_________________________________________________________________
dense_2 (Dense)              (None, 24, 3)             12
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 3)             0
_________________________________________________________________
dense_3 (Dense)              (None, 24, 17)            68
=================================================================
Total params: 419
Trainable params: 419
Non-trainable params: 0
```

Fig. 14. Arquitectura Autoencoder. Elaboración Propia.

Los resultados obtenidos desde el Autoencoder para la detección de anomalías se presentan a continuación:

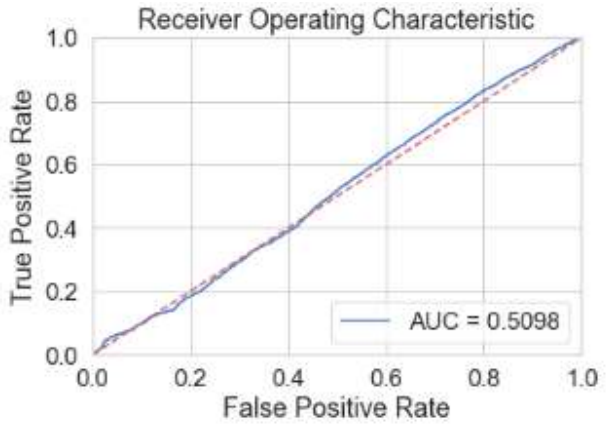

Fig. 15. Curva ROC. Elaboración Propia.

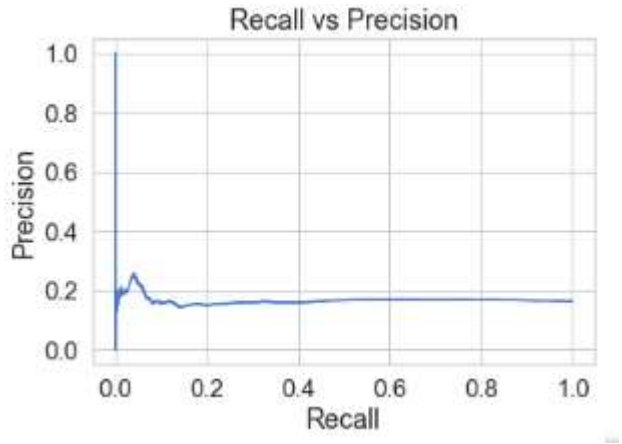

Fig 12: Recall para distintos valores de Threshold (Threshold = 0.9)

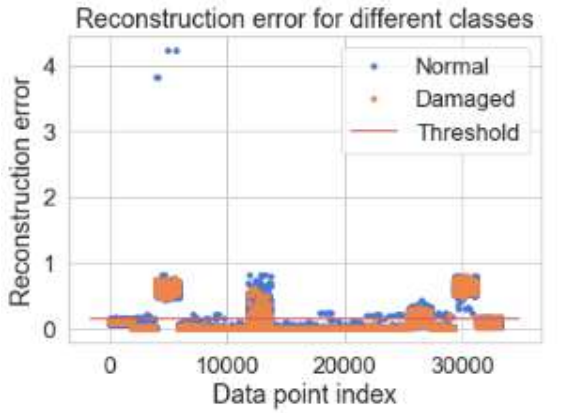

Fig. 16: Error de Reconstrucción para cada clase.

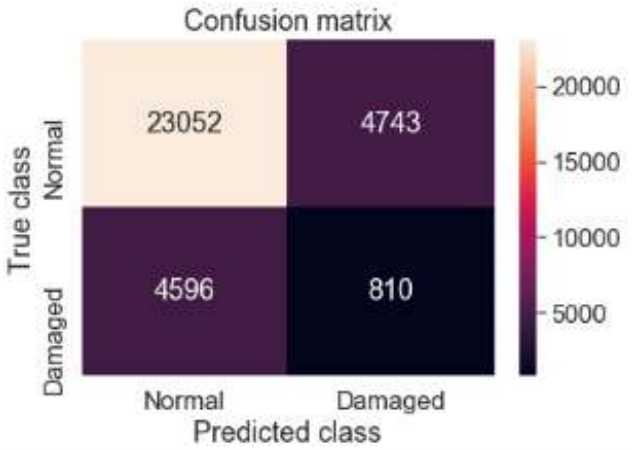

Fig. 17: Matriz de Confusión del detector de anomalías.

Para agregar Bayes, se agrega una capa Dense Flipout a la arquitectura de la red

Hiperparametros utilizados:

- Número de capas: 7.
- Batch size: 100.
- Número de épocas: 100.

Y la arquitectura de la red se ve de la siguiente forma:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 24, 16)            288
_________________________________________________________________
dropout (Dropout)            (None, 24, 16)            0
_________________________________________________________________
dense_1 (Dense)              (None, 24, 3)             51
_________________________________________________________________
dense_2 (Dense)              (None, 24, 3)             12
_________________________________________________________________
dropout_1 (Dropout)          (None, 24, 3)             0
_________________________________________________________________
dense_flipout (DenseFlipout) (None, 24, 17)            119
=================================================================
Total params: 470
Trainable params: 470
Non-trainable params: 0
```

Fig. 18: Arquitectura del Autoencoder con Bayes. Elaboración Propia.

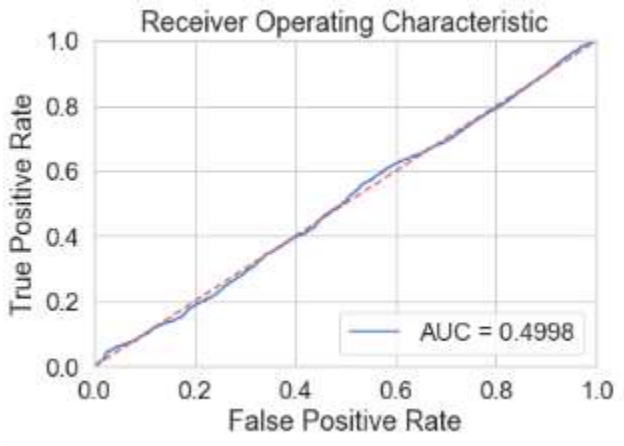

Fig. 19: Curva ROC Autoencoder Bayes.

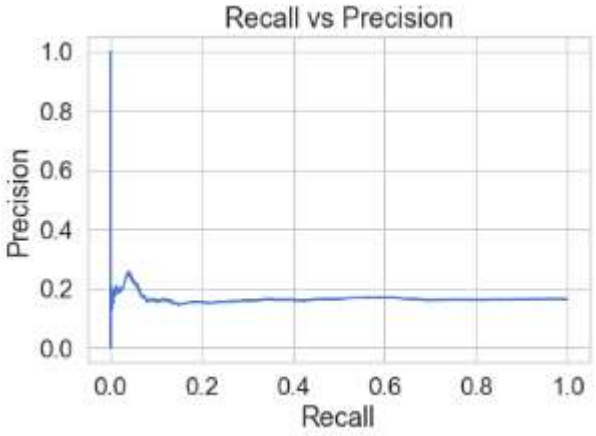

Fig. 20: Curva Precisión vs Recall Autoencoder Bayes.

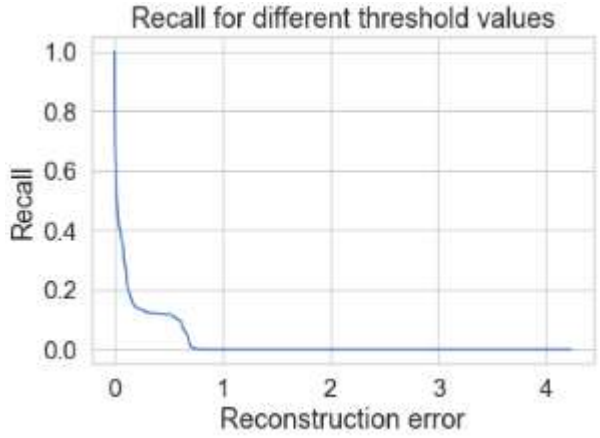

Fig. 21: Recall para distintos Thresholds.

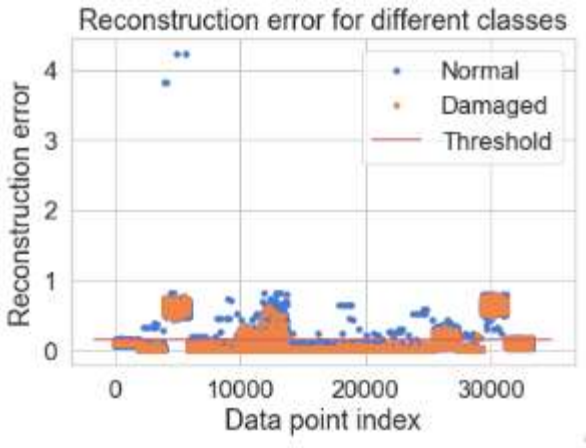

Fig. 22: Índice de puntos del error de reconstrucción.

Fig. 23: Matriz de Confusión, Autoencoder Bayes.

En base a los resultados obtenidos, se puede concluir que la adición de incertidumbre de Bayes resulta en que una gran cantidad de puntos que anteriormente estaban debajo del umbral que define una anomalía, ahora se encuentran por encima de este. Convirtiéndolos en puntos a analizar más en detalle con redes más refinadas para así asegurar la verdadera naturaleza de los puntos analizados. Lamentablemente no se pudo refinar más la red a tiempo por lo que los resultados no son tan notorios, pero se tiene una medida aproximada del valor del intervalo de incertidumbre en base a la diferencia de cantidades apreciables en las matrices de confusión:

Valor asociado a la incertidumbre = 9.6%

Diagnóstico de Fallas

El algoritmo del clasificador está construido con los siguientes parámetros y con la siguiente arquitectura:

Hiperparametros:

- Numero de Capas: 5.
- Tamaño de batch: 500.
- Número de épocas: 15
- 0.
- Número de neuronas en la capa de entrada: 64.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 64)                15936
_________________________________________________________________
flatten (Flatten)            (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 128)               8320
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 516
=================================================================
Total params: 24,772
Trainable params: 24,772
Non-trainable params: 0
```

Fig. 24: Arquitectura Clasificador.

Los resultados obtenidos desde el clasificador para el diagnóstico de fallas se presentan a continuación.

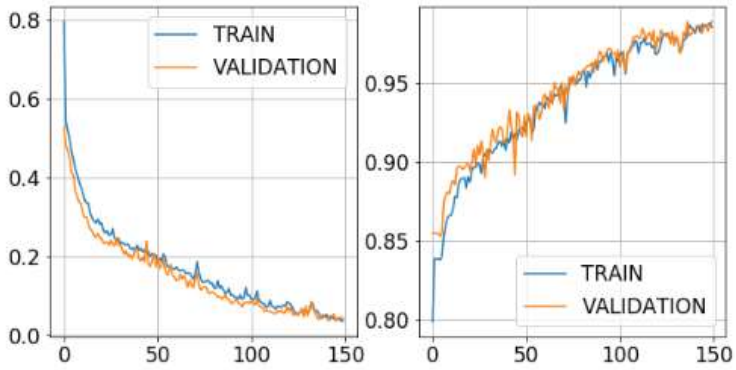

Fig. 25: Gráfico de validación y precisión.

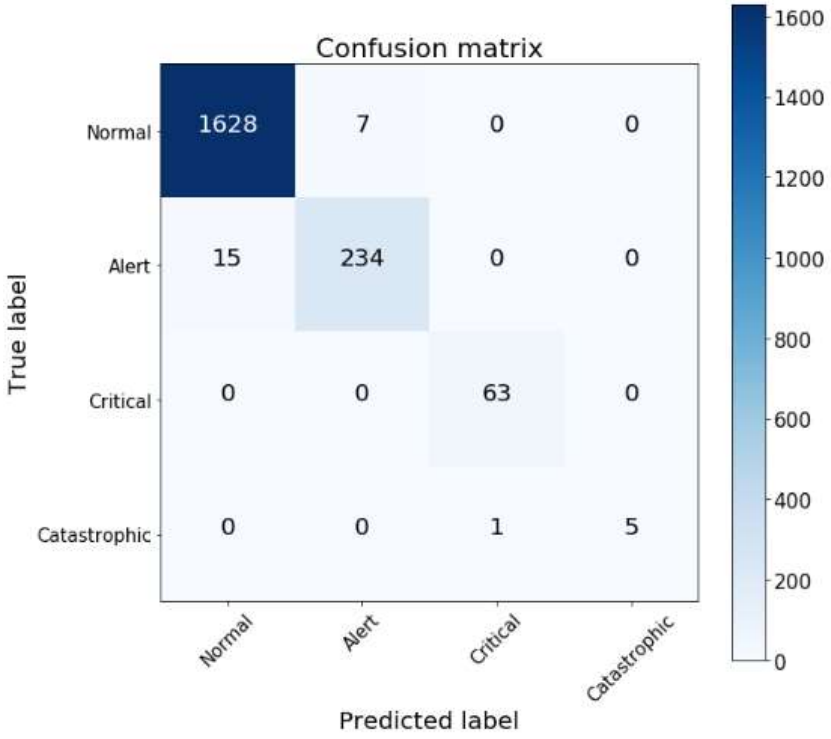

Fig. 26: Matriz de Confusión del clasificador.

Fig 18 y 19: Gráficos de validación y perdida/Matriz de Confusión del Clasificador.

Y después de agregar incertidumbre, manteniendo los mismos hiperparametros del caso regular, la arquitectura de la red se ve de la siguiente forma:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
gru (GRU)                    (None, 64)                15936
_________________________________________________________________
flatten (Flatten)            (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_flipout (DenseFlipout) (None, 128)               16512
_________________________________________________________________
dense (Dense)                (None, 4)                 516
=================================================================
Total params: 32,964
Trainable params: 32,964
Non-trainable params: 0
```

Fig. 27: Arquitectura red neuronal con Bayes

Los resultados son los siguientes:

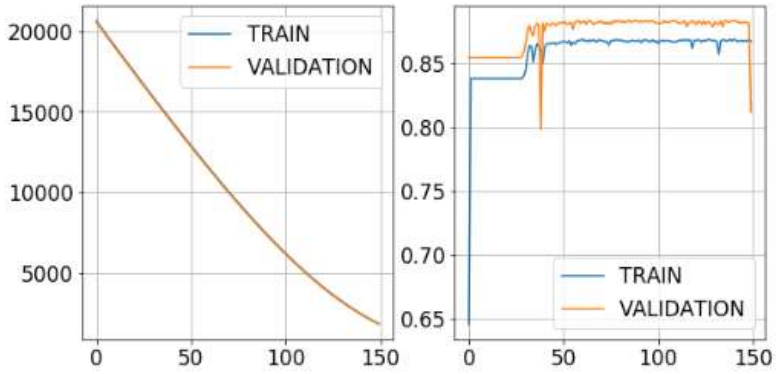

Fig. 28: Gráficos de validación y perdida.

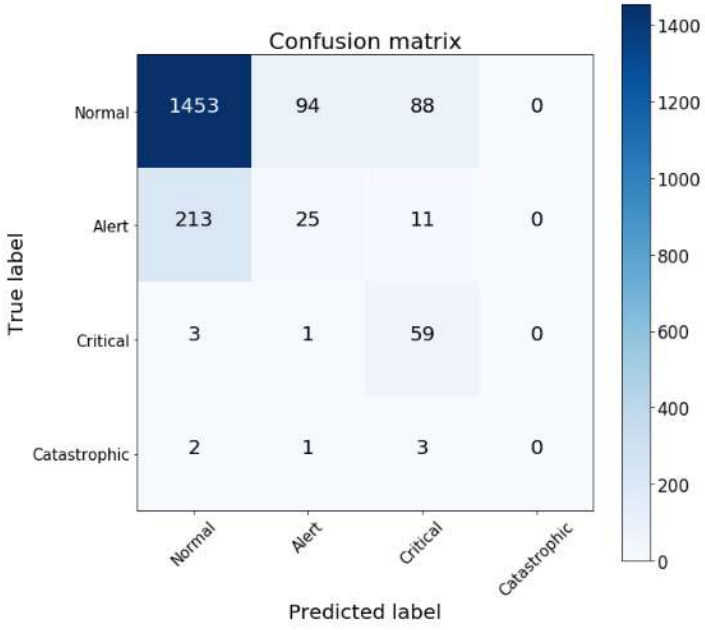

Fig. 29: Matriz de Confusión post Bayes.

De los gráficos obtenidos, lo que se puede observar es que la matriz de confusión contiene una mayor cantidad de falsos positivos y falsos negativos que la matriz obtenida con el clasificador regular. Lo primero que se puede apreciar es el impacto de agregar incertidumbre a la muestra, después, se puede notar que, dado el aumento del número de falsos positivos, la cantidad de

puntos a evaluar crece respecto al análisis normal, es decir, ahora podemos considerar cada falso positivo como un nuevo punto a evaluar ya que esto nos muestra que no hay certeza de que ese sensor vaya a fallar en el punto específico.

Conclusiones

Del presente trabajo de título, se aprende lo que es Deep Learning al punto de poder desarrollar redes neuronales propias, capaces de trabajar con los datos disponibles y que dan resultados adecuados al propósito de esta investigación.

Del aprendizaje conseguido durante este trabajo acerca del Deep Learning, se reconoce que, para procesos de detección de anomalías, los autoencoders son el mejor tipo de red neuronal para analizarlas, dado que son capaces de replicar los datos de entrada, permitiendo el análisis posterior de datos sucios.

Además, del aprendizaje se reconoce los clasificadores como la mejor forma de diagnosticar los niveles de falla del problema, gracias a que los datos recopilados tienen un formato con múltiples clases, un clasificador resuelve el problema de manera rápida y la matriz de confusión resulta una forma visualmente clara para realizar los análisis posteriores.

Del aprendizaje de redes bayesianas, se concluye que la manera de agregar incertidumbre a una red depende mucho del tipo de red utilizada y del conjunto de datos que se tiene. Dependiendo del tipo de red es el tipo de capa que hay que agregar de manera que los pesos y sesgos influyan para que haya una incertidumbre controlada dentro del sistema. Se utilizó una herramienta, la capa DenseFlipout, sobre las redes con el objetivo de que la red siga cumpliendo su rol, pero que los resultados obtenidos contengan una incertidumbre controlada y que refleje lo que se intenta comprobar.

Del análisis de los resultados de este trabajo de título, se concluye que las redes bayesianas construidas sirven como un buen complemento al análisis regular con redes neuronales, es decir, que no es un tipo de análisis que permite reemplazar el trabajo con redes neuronales de cualquier tipo, dado que permite revisar puntos que, en el análisis de una red regular, podrían salir como negativos de falla bajo conceptos convencionales. Un análisis con redes neuronales bayesianas posterior a uno normal podría resultar en mejores análisis de falla en máquinas como la bomba revisada en este trabajo, ya que los resultados detectarían puntos potencialmente propensos a fallar, y esto permite a los operadores de estas máquinas la habilidad de preparar mejores rutinas de mantenimiento predictivo en el futuro.

En los estudios realizados en el presente trabajo, se concluye que agregar incertidumbre a una red es un proceso que requiere un análisis concreto que justifique su uso, principalmente

porque no todo resultado obtenido de una red bayesiana tiene sentido inmediato y por ende requiere un procesamiento posterior que evalúe si valió la pena hacer el estudio. Que un trabajo de redes neuronales bayesianas es poderoso, pero no definitivo, dada la naturaleza aleatoria de agregar incertidumbre a sistemas, por lo que, en la opinión de este trabajo, se deben complementar con procesos empíricos no aleatorios y con análisis posteriores detallados.

Bibliografía

1. - Sayed, Sohail https://towardsdatascience.com/machine-learning-is-the-future-of-cancer-prediction; Visitado 18/5/2020

2. - Montgomery, Mike; https://www.forbes.com/sites/mikemontgomery/2019/06/05/self-driving-cars-are-already-here/ ; Visitado 18/5/2020

3. - Shridhar, Kumar; https://medium.com/neuralspace/bayesian-neural-network-series-post-2-background-knowledge-fdec6ac62d43; Visitado 18/5/2020

4. – Saha, Sumit; https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way; Visitado 18/5/2020

5. - Mullachery, Vikram; Khera, Aniruddh; Husain, Amir; “A study of Bayesian Neural Networks"

6.- Chopra, Paras; https://towardsdatascience.com/making-your-neural-network-say-i-dont-know-bayesian-nns-using-pyro-and-pytorch-b1c24e6ab8cd; Visitado 18/5/2020

7. - "TensorFlow: Open source machine learning" "It is machine learning software being used for various kinds of perceptual and language understanding tasks" — Jeffrey Dean, minute 0:47 / 2:17 from YouTube clip

8.- https://www.esp-richmond.com/blog/2017/12/19/8-tips-for-centrifugal-pump-safety Visitado 4/4/2020

9.- Wu, Motoki; https://towardsdatascience.com/adding-uncertainty-to-deep-learning-ecc2401f2013; Visitado 4/4/2020

10.- Laumann, Felix; https://towardsdatascience.com/when-machine-learning-meets-complexity-why-bayesian-deep-learning-is-unavoidable-55c97aa2a9cc; Visitado 10/5/2020

Anexos

Anexo A – Código para Detección de Anomalías

```
import tensorflow as tf
from tensorflow.keras import layers
import tensorflow_probability as tfp
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras

import math

# alterar graficos globalmente
plt.rcParams['font.size'] = 17

# metrics
from sklearn.metrics import r2_score
from sklearn.metrics import max_error
from sklearn.metrics import mean_squared_error

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from cm_plot import plot_confusion_matrix

sns.set(style='whitegrid', palette='muted', font_scale=1.5)

RANDOM_SEED = 42
LABELS = ["Normal", "Damaged"]

dataset = np.load('sync_data.npz')
x_train = dataset['x_train']
x_test = dataset['x_test']
y_train = dataset['y_train']
y_test = dataset['y_test']

batch_size = 128
epochs = 100
inChannel = 1
input_dim = 24
kernel = np.random.rand()
bias = np.random.rand()
print(kernel)
print(bias)
kernel = np.array(kernel)
bias = np.array(bias)
```

```
start = time.time()
model = tf.keras.Sequential()
#encoder
model.add(layers.Dense(units=16, activation = 'tanh', input_shape = (24,17)))
model.add(layers.Dropout(rate=0.01))
model.add(layers.Dense(units=3, activation = 'relu'))

#decoder
model.add(layers.Dense(units=3, activation = 'tanh'))
model.add(layers.Dropout(rate=0.01))
model.add(layers.Dense(units=17, activation = 'relu'))


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

print(model.summary())

model_history = model.fit(x_train, x_train, batch_size = 128, epochs = 100, validation_split = 0.2,
                          verbose = 1)
#%% Plot Plot the loss function history for the training and validation datasets

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()

#%%# Reconstruction error

predictions = model.predict(x_test)
mse = np.mean(np.power(x_test - predictions, 2), axis = 1)
mse = np.matrix.transpose(mse)
mse = np.reshape(mse,-1)
#print(mse)
y_test = np.matrix.transpose(y_test)
y_test = np.reshape(y_test,-1)
#print(y_test)
```

```
j = 0
while j < len(y_test):
    if 0 < y_test[j]:
        y_test[j] = 1
    j = j + 1
y_test0 = y_test
i = 0
while i < 16:
    y_test = np.concatenate((y_test, y_test0), axis = 0)
    i = i + 1
print(len(y_test))
#y_test = np.ones(33201, dtype = int)
error_df = pd.DataFrame({'reconstruction_error': mse, 'true_class': y_test})
error_df.describe()

#%%# Reconstruction error without damage

#fig = plt.figure()
#ax = fig.add_subplot(111)
normal_error_df = error_df[(error_df['true_class'] == 0) & (error_df['reconstruction_error'] < 1)]
#_ = ax.hist(normal_error_df.reconstruction_error.values, bins = 10)
#%%# Reconstruction error with damage

#fig = plt.figure()
#ax = fig.add_subplot(111)
damage_error_df = error_df[error_df['true_class'] == 1]
#_ = ax.hist(damage_error_df.reconstruction_error.values, bins = 10)

#%%# ROC

from sklearn.metrics import (confusion_matrix, precision_recall_curve, auc,
                             roc_curve, recall_score, classification_report, f1_score,
                             precision_recall_fscore_support)

print(error_df.true_class)
print(error_df.reconstruction_error)
```

```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, label='AUC = %0.4f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.001, 1])
plt.ylim([0, 1.001])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#%%# Precision and Recall
precision, recall, th = precision_recall_curve(error_df.true_class, error_df.reconstruction_error)
plt.plot(recall, precision, 'b', label='Precision-Recall curve')
plt.title('Recall vs Precision')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()

#%%
plt.plot(th, precision[1:], 'b', label='Threshold-Precision curve')
plt.title('Precision for different threshold values')
plt.xlabel('Threshold')
plt.ylabel('Precision')
plt.show()

#%%
plt.plot(th, recall[1:], 'b', label='Threshold-Recall curve')
plt.title('Recall for different threshold values')
plt.xlabel('Reconstruction error')
plt.ylabel('Recall')
plt.show()
```

```
threshold = np.mean(y_test)

groups = error_df.groupby('true_class')
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index, group.reconstruction_error, marker='o', ms=3.5, linestyle='',
            label= "Damaged" if name == 1 else "Normal")
ax.hlines(threshold, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", zorder=100, label='Threshold')
ax.legend()
plt.title("Reconstruction error for different classes")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();

#%%# Confusion Matrix
y_pred = [1 if e > threshold else 0 for e in error_df.reconstruction_error.values]
conf_matrix = confusion_matrix(error_df.true_class, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Anexo B – Código para Detección de Anomalías con Incertidumbre:

```
import tensorflow as tf
from tensorflow.keras import layers
import tensorflow_probability as tfp
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras

import math

# alterar graficos globalmente
plt.rcParams['font.size'] = 17

# metrics
from sklearn.metrics import r2_score
from sklearn.metrics import max_error
from sklearn.metrics import mean_squared_error

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from cm_plot import plot_confusion_matrix

sns.set(style='whitegrid', palette='muted', font_scale=1.5)

RANDOM_SEED = 42
LABELS = ["Normal", "Damaged"]

dataset = np.load('sync_data.npz')
x_train = dataset['x_train']
x_test = dataset['x_test']
y_train = dataset['y_train']
y_test = dataset['y_test']

batch_size = 128
epochs = 100
inChannel = 1
input_dim = 24
kernel = np.random.rand()
bias = np.random.rand()
print(kernel)
print(bias)
kernel = np.array(kernel)
bias = np.array(bias)
```

```
start = time.time()
model = tf.keras.Sequential()
#encoder
model.add(layers.Dense(units=16, activation = 'tanh', input_shape = (24,17)))
model.add(layers.Dropout(rate=0.01))
model.add(layers.Dense(units=3, activation = 'relu'))

#decoder
model.add(layers.Dense(units=3, activation = 'tanh'))
model.add(layers.Dropout(rate=0.01))
model.add(tfp.layers.DenseFlipout(units=17, activation = 'relu'))


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

print(model.summary())

model_history = model.fit(x_train, x_train, batch_size = 128, epochs = 100, validation_split = 0.2,
                          verbose = 1)
#%% Plot Plot the loss function history for the training and validation datasets

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()

#%%# Reconstruction error

predictions = model.predict(x_test)
mse = np.mean(np.power(x_test - predictions, 2), axis = 1)
mse = np.matrix.transpose(mse)
mse = np.reshape(mse,-1)
#print(mse)
y_test = np.matrix.transpose(y_test)
y_test = np.reshape(y_test,-1)
#print(y_test)
```

```
j = 0
while j < len(y_test):
    if 0 < y_test[j]:
        y_test[j] = 1
    j = j + 1
y_test0 = y_test
i = 0
while i < 16:
    y_test = np.concatenate((y_test, y_test0), axis = 0)
    i = i + 1
print(len(y_test))
#y_test = np.ones(33201, dtype = int)
error_df = pd.DataFrame({'reconstruction_error': mse, 'true_class': y_test})
error_df.describe()

#%%# Reconstruction error without damage

#fig = plt.figure()
#ax = fig.add_subplot(111)
normal_error_df = error_df[(error_df['true_class'] == 0) & (error_df['reconstruction_error'] < 1)]
#_ = ax.hist(normal_error_df.reconstruction_error.values, bins = 10)
#%%# Reconstruction error with damage

#fig = plt.figure()
#ax = fig.add_subplot(111)
damage_error_df = error_df[error_df['true_class'] == 1]
#_ = ax.hist(damage_error_df.reconstruction_error.values, bins = 10)

#%%# ROC

from sklearn.metrics import (confusion_matrix, precision_recall_curve, auc,
                             roc_curve, recall_score, classification_report, f1_score,
                             precision_recall_fscore_support)

print(error_df.true_class)
print(error_df.reconstruction_error)
```

```
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, label='AUC = %0.4f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.001, 1])
plt.ylim([0, 1.001])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()

#%%# Precision and Recall
precision, recall, th = precision_recall_curve(error_df.true_class, error_df.reconstruction_error)
plt.plot(recall, precision, 'b', label='Precision-Recall curve')
plt.title('Recall vs Precision')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.show()

#%%
plt.plot(th, precision[1:], 'b', label='Threshold-Precision curve')
plt.title('Precision for different threshold values')
plt.xlabel('Threshold')
plt.ylabel('Precision')
plt.show()

#%%
plt.plot(th, recall[1:], 'b', label='Threshold-Recall curve')
plt.title('Recall for different threshold values')
plt.xlabel('Reconstruction error')
plt.ylabel('Recall')
plt.show()
```

```
threshold = np.mean(y_test)

groups = error_df.groupby('true_class')
fig, ax = plt.subplots()

for name, group in groups:
    ax.plot(group.index, group.reconstruction_error, marker='o', ms=3.5, linestyle='',
            label= "Damaged" if name == 1 else "Normal")
ax.hlines(threshold, ax.get_xlim()[0], ax.get_xlim()[1], colors="r", zorder=100, label='Threshold')
ax.legend()
plt.title("Reconstruction error for different classes")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show();

#%%# Confusion Matrix
y_pred = [1 if e > threshold else 0 for e in error_df.reconstruction_error.values]
conf_matrix = confusion_matrix(error_df.true_class, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(conf_matrix, xticklabels=LABELS, yticklabels=LABELS, annot=True, fmt="d");
plt.title("Confusion matrix")
plt.ylabel('True class')
plt.xlabel('Predicted class')
plt.show()
```

Anexo C – Código para el Clasificador:

```
import tensorflow as tf
from tensorflow.keras import layers
import tensorflow_probability as tfp
import time
import numpy as np
import matplotlib.pyplot as plt
# alterar graficos globalmente
plt.rcParams['font.size'] = 17

# metrics
from sklearn.metrics import r2_score
from sklearn.metrics import max_error
from sklearn.metrics import mean_squared_error

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from cm_plot import plot_confusion_matrix
```

```
dataset = np.load('sync_data.npz')
x_train = dataset['x_train']
x_test = dataset['x_test']
y_train = dataset['y_train']
y_test = dataset['y_test']


#converting to binary matrices
y_bintrain = tf.keras.utils.to_categorical(y_train)
y_bintest = tf.keras.utils.to_categorical(y_test)


print('Data Loaded...')
####TRAINING


print('Initializing Model...')

# Initialize the LSTM and build the Graph.


start = time.time()
model = tf.keras.Sequential()
model.add(layers.GRU(units = 64, input_shape = (24,17)))
model.add(layers.Flatten())
model.add(layers.Dropout(rate = 0.05))
model.add(layers.Dense(units = 128 , activation = 'relu'))
model.add(layers.Dense(units = 4, activation = 'softmax'))

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

print(model.summary())

model_history = model.fit(x_train, y_bintrain, batch_size = 500, epochs = 150, validation_split = 0.1,
                          verbose = 1)

total_time = time.time()-start
```

```
### PREDICTIONS
y_pred_train = np.argmax(model.predict(x_train), axis = 1)
y_pred_test = np.argmax(model.predict(x_test), axis = 1)

### METRICS
y_train_not_ohe = np.argmax(y_bintrain, axis = 1)
y_test_not_ohe = np.argmax(y_bintest, axis = 1)
acc_train = (accuracy_score(y_train_not_ohe, y_pred_train))
acc_test = (accuracy_score(y_test_not_ohe, y_pred_test))
```

```
best_run = np.argmax(acc_test)


print('\nTRAIN')
print('ACC: {:.3f} '.format(acc_train))



print('\nTEST')
print('ACC: {:.3f} '.format(acc_test))

print('\nTIME')
print('Time: {:.3f}  [min]'.format(total_time/60))


### PLOTS
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']
acc = model_history.history['accuracy']
val_acc = model_history.history['val_accuracy']



plt.figure(figsize = (10,5))
# plot1
plt.subplot(1,2,1)
plt.plot(loss, label = 'TRAIN')
plt.plot(val_loss, label = 'VALIDATION')
plt.legend()
plt.grid()
# plot2
plt.subplot(1,2,2)
plt.plot(acc, label = 'TRAIN')
plt.plot(val_acc, label = 'VALIDATION')
plt.legend()
plt.grid()
plt.show()
```

Anexo D – Código para el Clasificador con Incertidumbre:

```
import tensorflow as tf
from tensorflow.keras import layers
import tensorflow_probability as tfp
import time
import numpy as np
import matplotlib.pyplot as plt
# alterar graficos globalmente
plt.rcParams['font.size'] = 17

# metrics
from sklearn.metrics import r2_score
from sklearn.metrics import max_error
from sklearn.metrics import mean_squared_error

from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from cm_plot import plot_confusion_matrix
```

```
dataset = np.load('sync_data.npz')
x_train = dataset['x_train']
x_test = dataset['x_test']
y_train = dataset['y_train']
y_test = dataset['y_test']


#converting to binary matrices
y_bintrain = tf.keras.utils.to_categorical(y_train)
y_bintest = tf.keras.utils.to_categorical(y_test)


print('Data Loaded...')
####TRAINING


print('Initializing Model...')

# Initialize the LSTM and build the Graph.


start = time.time()
model = tf.keras.Sequential()
model.add(layers.GRU(units = 64, input_shape = (24,17)))
model.add(layers.Flatten())
model.add(layers.Dropout(rate = 0.05))
model.add(tfp.layers.DenseFlipout(units = 128 , activation = 'relu'))
model.add(layers.Dense(units = 4, activation = 'softmax'))

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

print(model.summary())

model_history = model.fit(x_train, y_bintrain, batch_size = 500, epochs = 150, validation_split = 0.1,
                          verbose = 1)

total_time = time.time()-start
```

```
### PREDICTIONS
y_pred_train = np.argmax(model.predict(x_train), axis = 1)
y_pred_test = np.argmax(model.predict(x_test), axis = 1)

### METRICS
y_train_not_ohe = np.argmax(y_bintrain, axis = 1)
y_test_not_ohe = np.argmax(y_bintest, axis = 1)
acc_train = (accuracy_score(y_train_not_ohe, y_pred_train))
acc_test = (accuracy_score(y_test_not_ohe, y_pred_test))
```

```
best_run = np.argmax(acc_test)


print('\nTRAIN')
print('ACC: {:.3f} '.format(acc_train))



print('\nTEST')
print('ACC: {:.3f} '.format(acc_test))

print('\nTIME')
print('Time: {:.3f}  [min]'.format(total_time/60))


### PLOTS
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']
acc = model_history.history['accuracy']
val_acc = model_history.history['val_accuracy']



plt.figure(figsize = (10,5))
# plot1
plt.subplot(1,2,1)
plt.plot(loss, label = 'TRAIN')
plt.plot(val_loss, label = 'VALIDATION')
plt.legend()
plt.grid()
# plot2
plt.subplot(1,2,2)
plt.plot(acc, label = 'TRAIN')
plt.plot(val_acc, label = 'VALIDATION')
plt.legend()
plt.grid()
plt.show()
```