MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

CONSTRUCCIÓN DE UN SISTEMA DE CONTROL DE TEMPERATURA EN LA INDUSTRIA 4.0 CON SISTEMAS DE BAJA POTENCIA

Autor: Manuel Vázquez Oliva
Director: Federico Muñoz Babiano

Madrid

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Construcción de un sistema de control de temperatura en la industria 4.0 con sistemas de baja potencia

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2023/24 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Manuel Vázquez Oliva Fecha: 11/ 07/ 2024

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Federico Muñoz Babiano Fecha: 11/ 07/ 2024

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

CONSTRUCCIÓN DE UN SISTEMA DE CONTROL DE TEMPERATURA EN LA INDUSTRIA 4.0 CON SISTEMAS DE BAJA POTENCIA

Autor: Manuel Vázquez Oliva
Director: Federico Muñoz Babiano

Madrid

Agradecimientos

A mis padres por permitirme realizar este máster y haberme apoyado en los momentos complicados.

A Cristina por ser un apoyo incondicional.

CONSTRUCCIÓN DE UN SISTEMA DE CONTROL DE TEMPERATURA EN LA INDUSTRIA 4.0 CON SISTEMAS DE BAJA POTENCIA

Autor: Vázquez Oliva, Manuel.
Director: Muñoz Babiano, Federico.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

El presente trabajo busca desarrollar un sistema de mantenimiento predictivo de motores eléctricos, empleando para ello la inteligencia artificial y empleando como parámetro de análisis la temperatura. La necesidad de la industria en prevenir fallos en los sistemas de producción es lo que lo motiva.

Los métodos tradicionales para realizar estos mantenimientos se centran en el análisis de señales y en el diagnóstico de las vibraciones de los dispositivos; mostrando algunas limitaciones que se solventarían aplicando la termografía al ser este un método no invasivo.

Para poder construir este sistema, lo primero que se debe hacer es programar una red neuronal que sea capaz de diferenciar los motores en base a las imágenes térmicas que se obtengan. Se decide implementar una CNN programada en *Python* con *Tensorflow* y *Keras*. Para poder entrenarla se hace uso de un *dataset* que incluye imágenes térmicas de motores funcionando correctamente, motores que están desalineados y motores que tienen algunas jaulas rotas. Para poder partir de un modelo sencillo, primero se entrenó una red que no emplea el aumento de datos y una vez obtenidos resultados óptimos se incluyó esta técnica para hacerla más fiable.

Los resultados obtenidos con distintas métricas muestran que la red con aumento de datos es mucho mejor y además es funcional, aunque se podría seguir mejorando. Para ello se deberían entrenar modelos variando distintas métricas como pueden ser las de las capas de convolución.

En conclusión, este trabajo propone una parte fundamental del sistema de control de temperatura de los motores como es la red que analizaría las imágenes y ayudaría a realizar ese mantenimiento predictivo que ayudaría a reducir los costes y a mejorar la eficiencia energética.

1. Introducción

En la industria actual, la detección y diagnóstico predictivo de fallos en los sistemas de producción son fundamentales. El crecimiento de la industria 4.0 ha ocasionado una tendencia hacia predecir posibles fallos o errores en los distintos eslabones de la cadena de producción. Los motores de inducción constituyen una parte sustancial de la misma ya que se emplean en el movimiento de ventiladores, bombas, compresores, etc. El uso continuado hace que se vayan deteriorando y si ese deterioro no es identificado a tiempo, se traduce en unos costes que podrían haberse evitado.

Es por ello por lo que el concepto de monitorización se introduce en la industria, reduciendo los costes relacionados con las reparaciones y minimizando el tiempo de paradas en las máquinas. Todo esto va de la mano de conseguir desarrollar tecnologías que ayuden a reducir el impacto medioambiental de las operaciones industriales, ya que al realizar un mantenimiento predictivo se aumenta la eficiencia energética al funcionar los equipos de manera óptima y se prolonga la vida útil de los mismos al ser las reparaciones significativas mucho menores (T. Orlowska-Kowalska *et al,2022).*

Actualmente, las estrategias más usadas en la detección de anomalías son el análisis de señales y el diagnóstico de vibraciones mecánicas. Estos métodos presentan algunos inconvenientes como pueden ser las conexiones mecánicas y eléctricas y los recursos computacionales. Debido a estos problemas, cada vez más se prefiere emplear métodos no invasivos, aunque presenten también algunos inconvenientes. En el ámbito industrial la termografía infrarroja es la técnica más popular actualmente. Las principales ventajas que presenta son: ausencia de impacto en el dispositivo observado, eficiencia en la monitorización y la portabilidad y simplicidad operática.

2. Definición del proyecto

Este trabajo se centra en el desarrollo y la implementación de un sistema de mantenimiento predictivo para motores eléctricos. A partir de técnicas de inteligencia artificial se pretende poder identificar si un dispositivo está funcionando correctamente o si por el contrario presenta algún fallo.

Es por ello por lo que se ha diseñado una red neuronal convolucional (CNN) que ha sido programada en Python y que a partir de imágenes térmicas es capaz de identificar si un motor está funcionando correctamente, si está desalineado o si tiene alguna de sus jaulas rotas. Se ha decidido emplear este tipo de red por su por su eficacia en tareas de reconocimiento de patrones y clasificación de imágenes.

El sistema trata de mejorar la eficiencia y la fiabilidad en el mantenimiento de este tipo de máquinas realizando un mantenimiento predictivo de los motores.

3. Descripción del modelo

Mediante el uso de herramientas de inteligencia artificial como *Keras* y *Tensorflow* se construye una red neuronal convolucional. Usando como *dataset* imágenes térmicas de motores de inducción tomadas en condiciones controladas, se entrena el modelo hasta llegar a obtener resultados fiables.

Las imágenes fueron clasificadas en los tres distintos casos que presenta el *dataset* y se convirtieron a blanco y negro para así permitir a la red procesarlas de manera más eficaz. Además, se dividieron los datos en tres sets: entrenamiento, validación y test.

Para la configuración de la red neuronal convolucional, se emplearon capas de convolución junto con capas de agrupación. Además, se han programado dos modelos, uno que emplea la técnica de aumento de datos y otro que no la emplea.

El modelo que no emplea esta técnica está compuesto por 4 capas de convolución de 32, 64, 128 y 256 núcleos. A su vez se incluye una capa de *Dropout*, para evitar el sobreajuste, en la que se desconectan el 50% de los núcleos. Por último, se incluye una capa densa con 512 núcleos con una función de activación ReLU. La capa de salida es de 3 núcleos que se corresponden con cada una de las tres clases del estado del motor y emplea una función de activación *softmax*.

La red neuronal que emplea el aumento de datos tiene como diferencias en su estructura el empleo de tres técnicas para evitar el sobreajuste que se producía durante el entrenamiento de este modelo. Se define una tasa de aprendizaje del optimizador Adam para así conseguir que el ajuste sea más preciso. También se incluye la técnica conocida como *early stopping* con la que se para el entrenamiento cuando la red ya no está aprendiendo más. Por último, se aplica la regularización L2 a la función de perdida para penalizar los pesos más grandes. A parte de estos cambios, también se modifica el tamaño de las imágenes que se introducen al modelo y el número de imágenes en cada época de entrenamiento.

4. Resultados

Tras realizar pruebas con distintos modelos, finalmente se obtuvieron los siguientes resultados con los datos de test modificados al aplicarles el aumento de datos:

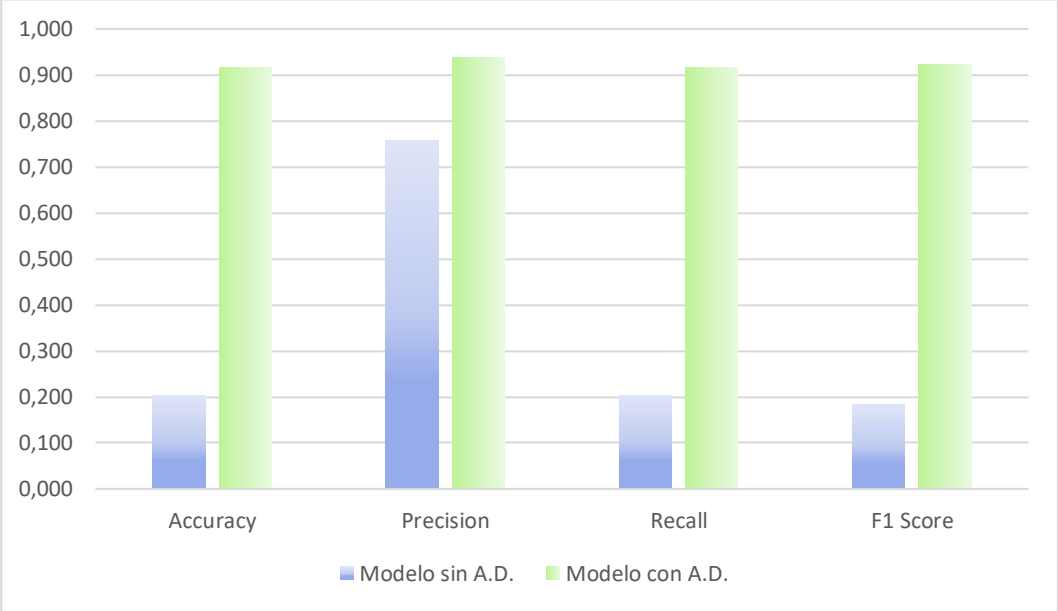

Ilustración 1. Comparativa de métricas para los dos modelos programados

En la imagen anterior se puede observar como el modelo con aumento de datos se comporta mucho mejor que el que no lo tiene. Analizando de manera individual cada métrica:

- *Accuracy:* El modelo con A.D. obtiene un 92% lo que nos indica una mayor robustez y que actuará mejor ante variaciones de los datos.
- *Precision:* Aunque el modelo sin A.D obtiene un buen valor, el otro modelo siendo superior indicándonos que esta red es capaz de identificar mejor los casos que son falsos positivos.
- *Recall:* La red neuronal que se comporta de mejor manera obtiene un 91,7% esta métrica nos indica que identifica mejor los casos que son realmente positivos.
- *F1 score:* Esta métrica nos muestra el equilibrio entre *precision* y *recall* siendo como en el resto de los parámetros mejor la red con el aumento de datos. Esto nos indica que es mucho más confiable.

5. Conclusiones

Este trabajo muestra parte de un sistema para el mantenimiento predictivo de manera no invasiva. Queda demostrado que, con el uso de la inteligencia artificial se podrían ahorrar muchos costes a la par que se podría mejorar la eficiencia energética. Se puede observar que el empleo de las redes neuronales convolucionales se justifica para tareas de reconocimiento de patrones. A su vez, la técnica de aumento de datos surge como una necesidad a la hora de analizar problemas complejos en los que los datos pueden ser presentados de diversas maneras. Por tanto, esta técnica permite generalizar los datos dotando a la red de una robustez que es necesaria en el análisis de imágenes. Por último,

la termografía que ha sido empleada en la obtención de las imágenes nos permite aplicar métodos no invasivos en la industria con las consiguientes ventajas que estos nos aportan a la hora de realizar los mantenimientos predictivos.

6. Referencias

[1] T. Orlowska-Kowalska *et al.*, "Fault Diagnosis and Fault-Tolerant Control of PMSM Drives–State of the Art and Future Challenges," in *IEEE Access*, vol. 10, pp. 59979-60024, 2022, doi: 10.1109/ACCESS.2022.3180153.

CONSTRUCTION OF A TEMPERATURE CONTROL SYSTEM IN INDUSTRY 4.0 WITH LOW POWER SYSTEMS.

Author: Vázquez Oliva, Manuel.
Supervisor: Muñoz Babiano, Federico.
Collaborating Entity: ICAI - Universidad Pontificia Comillas.

ABSTRACT

This work aims to develop a predictive maintenance system for electric motors, using artificial intelligence and temperature as an analysis parameter. It is motivated by industry's need to prevent failures in production systems.

Traditional methods to perform this maintenance focus on signal analysis and diagnosis of device vibrations; showing some limitations that would be solved by applying thermography as this is a non-invasive method.

To build this system, the first thing to do is to program a neural network capable of differentiating the engines based on the thermal images obtained. It was decided to implement a CNN programmed in Python with TensorFlow and Keras. To train it, use is made of a dataset that includes thermal images of engines that are working correctly, engines that are misaligned and engines that have some broken cages. In order to start from a simple model, a network that does not use data augmentation was first trained and once optimal results were obtained, this technique was included to make it more reliable.

The results obtained with different metrics show that the data augmented network is much better and functional, although it could be further improved. To do so, models should be trained by varying different metrics such as convolution layers.

In conclusion, this work proposes a fundamental part of the motor temperature control system as the network that would analyse the images and help to carry out predictive maintenance that would help to reduce costs and improve energy efficiency.

1. Introduction

In today's industry, the detection and predictive diagnosis of faults in production systems is essential. The growth of Industry 4.0 has led to a trend towards predicting possible failures or errors in the different links of the production chain. Induction motors are a substantial part of this as they are used in the movement of fans, pumps, compressors, etc. Continuous use causes them to deteriorate and if this deterioration is not identified in time, it results in costs that could have been avoided.

This is why the concept of monitoring is being introduced in industry, reducing repair-related costs and minimising machine downtime. This goes hand in hand with developing technologies that help to reduce the environmental impact of industrial operations, as predictive maintenance increases energy efficiency by making equipment work optimally and prolongs the useful life of equipment as significant repairs are much less frequent (T. Orlowska-Kowalska et al,2022).

Currently, the most used strategies for anomaly detection are signal analysis and mechanical vibration diagnostics. These methods have some drawbacks such as mechanical and electrical connections and computational resources. Due to these problems, non-invasive methods are increasingly preferred, although they also have some drawbacks. In the industrial field, infrared thermography is currently the most popular technique. The main advantages are no impact on the observed device, monitoring efficiency, portability and operational simplicity.

2. Project definition

This work focuses on the development and implementation of a predictive maintenance system for electric motors. Using artificial intelligence techniques, the aim is to be able to identify whether a device is working correctly or whether it has a fault.

For this reason, a convolutional neural network (CNN) has been designed and programmed in Python which, based on thermal images, can identify whether a motor is working correctly, whether it is misaligned or whether one of its cages is broken. It has been decided to use this type of network because of its effectiveness in pattern recognition and image classification tasks.

The system aims to improve efficiency and reliability in the maintenance of this type of machine by performing predictive maintenance on the engines.

3. Description of the model

Using artificial intelligence tools such as Keras and Tensorflow, a convolutional neural network is built. Using as a dataset thermal images of induction motors taken under controlled conditions, the model is trained until reliable results are obtained.

The images were classified into the three different cases presented by the dataset and converted to black and white to allow the network to process them more efficiently. The data was further divided into three sets: training, validation and test.

For the configuration of the convolutional neural network, convolution layers were used together with clustering layers. In addition, two models have been programmed, one using the data augmentation technique and one not using the data augmentation technique.

The non-data augmentation model is composed of 4 convolution layers of 32, 64, 128 and 256 cores. In turn, a Dropout layer is included, to avoid overfitting, in which 50% of the cores are disconnected. Finally, a dense layer with 512 cores with a ReLU activation function is included. The output layer is 3 cores corresponding to each of the three classes of the engine state and employs a softmax activation function.

The neural network that uses data augmentation differs in its structure by using three techniques to avoid the over-fitting that occurred during the training of this model. A learning rate of the Adam optimiser is defined in order to make the adjustment more accurate. Also included is the technique known as early stopping, which stops training when the network is no longer learning. Finally, L2 regularisation is applied to the loss function to penalise larger weights. In addition to these changes, the size of the images that are introduced to the model and the number of images in each training epoch are also modified.

4. Results

After testing different models, the following results were finally obtained with the modified test data when the data enhancement was applied:

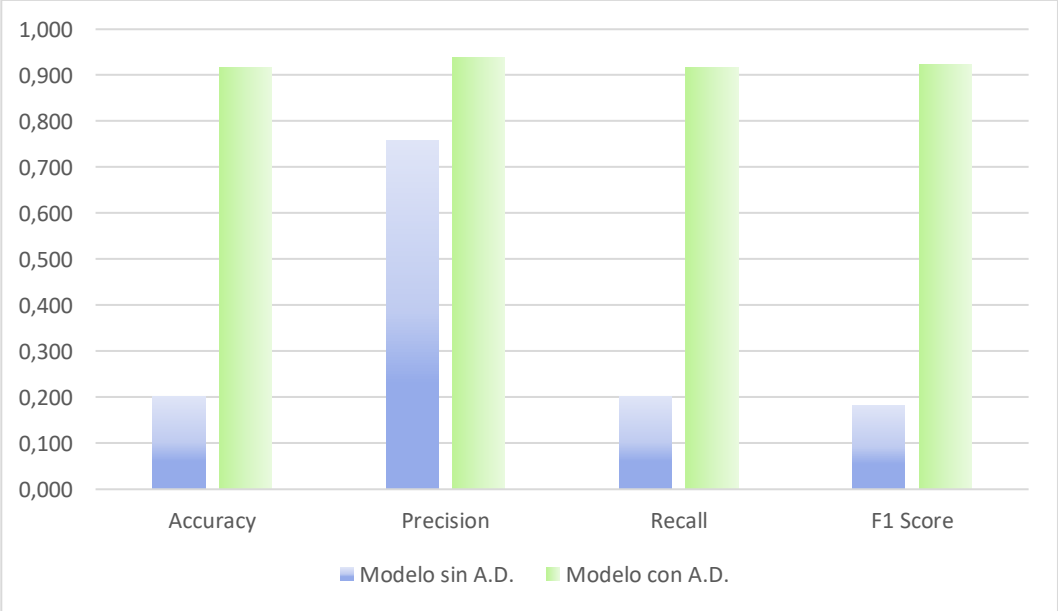

Illustration 1. Comparison of metrics for the two programmed models.

In the image above it can be seen how the model with data augmentation performs much better than the one without. Analysing each metric individually:

- *Accuracy*: The model with D.A. obtains a 92% which indicates a greater robustness and that it will perform better in the face of data variations.
- *Precision*: Although the model without D.A. obtains a good value, the other model is superior, indicating that this network is better able to identify false positive cases.
- *Recall*: The neural network that performs better obtains a 91.7%. This metric indicates that it identifies better the cases that are really positive.
- *F1 score*: This metric shows us the balance between precision and recall being as in the rest of the parameters better the network with the increase of data. This indicates that it is much more reliable.

5. Conclusions

This work shows part of a system for non-invasive predictive maintenance. It is shown that, with the use of artificial intelligence, many costs could be saved, and energy efficiency could be improved. The use of convolutional neural networks is justified for pattern recognition tasks. In turn, the data augmentation technique arises as a necessity when analysing complex problems where data can be presented in different ways. Therefore, this technique allows the data to be generalised, giving the network a robustness that is necessary in image analysis. Finally, thermography, which has been used to obtain the images, allows us to apply non-invasive methods in industry with the consequent advantages that they provide when carrying out predictive maintenance.

6. References

[1] T. Orlowska-Kowalska *et al.*, "Fault Diagnosis and Fault-Tolerant Control of PMSM Drives–State of the Art and Future Challenges," in *IEEE Access*, vol. 10, pp. 59979-60024, 2022, doi: 10.1109/ACCESS.2022.3180153.

Índice de la memoria

Índice de ilustraciones

Índice de tablas

Capítulo 1. INTRODUCCIÓN

En la actualidad, la interconexión y automatización de los procesos industriales han alcanzado un nivel de importancia muy elevado, dando lugar a la llamada Industria 4.0. Uno de los elementos más relevantes en la eficiencia de los procesos de fabricación es la temperatura, cuyo control es crucial. En muchas aplicaciones, este parámetro es decisivo para la calidad del producto final, la eficiencia del proceso y en definitiva la rentabilidad de la empresa. Además, cada vez es más común el mantenimiento predictivo de los distintos eslabones de la cadena de producción. En este contexto, los motores eléctricos constituyen una parte fundamental en este ámbito al ser quienes mueven las distintas bombas, ventiladores, etc.

El presente trabajo se enfoca en la construcción de un sistema de control de temperatura integrado en el contexto de la Industria 4.0 que además empleará un sistema de baja potencia. En concreto se van a utilizar imágenes térmicas, lo que constituirá un método no invasivo de medición de este parámetro aportando una serie de ventajas claves a la hora de realizar el mantenimiento de los equipos. Al desarrollar el proyecto incluyendo dichos sistemas de baja potencia, se busca responder a la creciente necesidad de optimizar el consumo de energía y reducir el impacto ambiental de las operaciones industriales sin olvidarse de la calidad del producto y la eficiencia.

Conforme continúa creciendo la demanda de tecnologías más sostenibles, el desarrollo de sistemas de control de temperatura que funcionen con un bajo consumo de energía se convierte en un objetivo estratégico en la industria. Este proyecto no solo tiene el propósito de cumplir con los estándares medioambientales y regulatorios cada vez más rigurosos; sino también proporcionar soluciones innovadoras que potencien la competitividad y sostenibilidad empresarial en el actual escenario industrial.

Por lo tanto, lo que se pretende a lo largo del siguiente documento es encontrar una solución que contribuya al avance de la Industria 4.0 con un sistema innovador y sostenible que permita reducir el impacto medioambiental a la par que aumente la eficiencia en sistemas de producción.

Capítulo 2. ESTADO DEL ARTE

2.1 DEFINICIÓN

En el presente trabajo se va a construir un sistema de control de temperatura de baja potencia para la industria 4.0. Por ello, se va a explicar brevemente en qué consisten y como están compuestos estos tipos de sistemas, así como en que consiste la industria 4.0 y que sistemas de baja potencia se pueden usar en la actualidad.

La Industria 4.0 se puede entender como la incorporación de las nuevas tecnologías a la producción, ya sea mediante la incorporación de robots que interactúan con los humanos, máquinas que hablan entre ellas o sistemas que pueden predecir un fallo antes de que ocurra; incluso cadenas de producción que se pueden adaptar a la demanda. A raíz de estas mejoras se va a conseguir recoger datos para analizarlos y así poder ser en la fabricación más rápidos, más eficientes y flexibles; permitiendo esto disminuir los costes y aumentar la calidad de los productos. La idea fundamental es que usando tecnologías emergentes se implemente el Internet de las Cosas (IoT) consiguiendo así los objetivos citados anteriormente (Peralta Abarca et al., 2020).

Los principales aspectos que abarca la industria 4.0 son los siguientes (Mababu Mukiur, 2022):

1. Personalización masiva de productos para satisfacer necesidades individuales
2. Adaptación automatizada de los procesos de producción para adaptarse a un entorno dinámico.
3. Modelos avanzados de interacción hombre-máquina, incluyendo maneras innovadoras de operar en las fábricas.
4. Control y autoconocimiento de los componentes individuales y su comunicación con otras máquinas.

En cuanto a los sistemas de baja potencia y alto alcance, se les conoce como LPWAN (*Low-Power Wide-Area Networking*). LPWAN se presenta como la mejor tecnología para el IoT en situaciones donde alta autonomía, bajo costo, largo alcance y además se vayan a transferir pequeñas cantidades de datos; sea el objetivo. Dentro de estos sistemas nos encontramos que las dos tecnologías más populares son LoRa y Sigfox(Rubio Aparicio, 2019).

Una opción para el presente trabajo sería emplear LoRa como capa física y LoRaWAN como capa de enlace, al ser una propuesta de código abierto. LoRa es una

solución de redes LPWAN que se basa en técnicas de transmisión de espectro ensanchado y modulación CSS. Las ventajas que presenta son (Pérez & Risco Ramos, 2020):

- Bajo consumo de energía.
- Largo alcance de transmisión.
- Posibilidad de contar con un servicio gratuito.
- Niveles de seguridad que ofrece.
- Bidireccionalidad.
- Incremento del rendimiento al facilitar la toma de decisiones oportuna basada en datos más precisos.
- Reducción de costos logísticos.

Los sistemas de control de temperatura están presentes en muchas aplicaciones industriales actualmente y también de investigación. En una amplia gama de estos ámbitos, mantener una temperatura constante, precisa y controlada; permite aumentar el rendimiento y calidad de la cadena.

La metodología más común en el control de temperatura es el control PID (Proporcional, Integral, Derivativo). A partir de tres términos: término proporcional, que nos va a devolver a las desviaciones de temperatura una respuesta instantánea, término integral, que va corrigiendo los errores que se acumulan a lo largo del tiempo y término derivativo, es el encargado de anticipar la querencia a cambiar de la temperatura; se controla de manera estable y exacta la temperatura.

Otro método es el predictivo. Se centra más en estimar comportamientos posteriores en vez de presentes y ajusta la temperatura en base a esa predicción. Es muy empleado para sistemas en los que la temperatura no se comporta de manera lineal o de formas muy complejas.

En la actualidad, los sistemas de medición de temperatura se emplean en diversas industrias, siendo algunas: industria energética, industria farmacéutica, industria química o industria alimentaria.

2.2 HISTORIA Y EVOLUCIÓN

En este capítulo se va a tratar de hacer un resumen de cómo han evolucionado los sistemas de control de temperatura y los sensores que se emplean para medir la misma (Picquart & Carrasco Morales, 2017).

Galileo, a principios del siglo XVII, mientras realizaba un experimento a cerca del calor, comprendió que el aire se contrae al enfriarse y se dilata al calentarse. Por este hecho se suele considerar a Galileo como el inventor del primer instrumento de medición

de la temperatura. Sin embargo, parece ser que la idea del termómetro de aire fue retomada por Santorre Santorio, quién inventó el primer aparato con graduación y sensible a la temperatura. Lo empleaba para medir la fiebre de sus pacientes y lo llamó termoscopio. El termómetro consta de tres grados: temperatura del aire enfriado por la nieve, aire calentado por la llama de una vela y punto medio.

No obstante, este tipo de aparato presenta un gran inconveniente que es que es sensible a los cambios de presión atmosférica. Fue Pascal quien demuestra este defecto de los termoscopios. A partir de entonces, se comienzan a desarrollar los termómetros

Ilustración 1. Termoscopio de Sartorio (Picquart & Carrasco Morales, 2017)

Posteriormente, se inventa el termómetro de vidrio que ya permitiría identificar las variaciones de temperatura, así como diferenciar entre calor y frío. Uno de los artilugios de esta índole, que usaban tanto el principio de Arquímedes, así como la dilatación de la materia para medir los cambios de temperatura, más importante es el conocido como termoscopio “de Galileo”. Uno de los mayores avances se produce a mediados del siglo XVII con la aparición de termómetros de vidrio cerrados que contenían un líquido dentro. Los más famosos fueron los termómetros de Florencia.

Ilustración 2. Termómetro de Florencia (Picquart & Carrasco Morales, 2017)

Ya en el siglo XIX se inventó el sensor de temperatura bimetálico. Su fundamento físico es la expansión desigual de dos placas de metal que están unidas. A partir de una curva, que se obtiene con los cambios de temperatura, es posible activar un termostato. No son los sensores con mayor precisión, pero su principal ventaja es el bajo precio que presentan.

En ese mismo siglo, se empezó a tener un gran interés por el estudio de la electricidad y todos los efectos asociados a la misma. Los investigadores descubrieron que se podrían emplear metales con distinta resistencia y conductividad para medir la temperatura. Tras distintos descubrimientos con distintos metales, Becquerel propuso emplear un termopar platino-platino basado en el fenómeno descubierto por Johann Seebeck años antes. Leopoldo Nobili fue quien lo llevó a cabo. Una aplicación del platino en otros sensores de temperatura es en los RTD, inventado en 1932. Este es considerado uno de los sensores más precisos para medir la temperatura.

Ilustración 3. Sensor de temperatura RTD (Lizán Ortiz et al., 2018)

Sir Frederick William Herschel descubrió la radiación infrarroja en 1800. En torno 1880 Samuel P. Langley creó el primer detector de radiación infrarroja, el cual era capaz de identificar la radiación debido al aumento de temperatura en un cuerpo absorbente de calor.

Es ya en 1958, después de grandes avances en esta tecnología durante la Segunda Guerra Mundial, cuando la empresa AGA (actual FLIR Systems) inventa la primera cámara térmica para empleo militar. Debido a la crisis energética de los años 70, se intentaba reducir las pérdidas de calor en los edificios y fue por ello por lo que se comenzó a implantar la termografía. Durante este periodo también se inventó el sensor piroeléctrico, cuyo funcionamiento también se basa en la detección de la radiación infrarroja.

Estos primeros dispositivos eran pesados, de gran tamaño y difíciles de manejar. Además, necesitaban de un sistema de refrigeración aparte por las temperaturas que alcanzaban. No fue hasta la década de los 90 cuando esta tecnología avanzó más y se dejaron de necesitar estos sistemas.

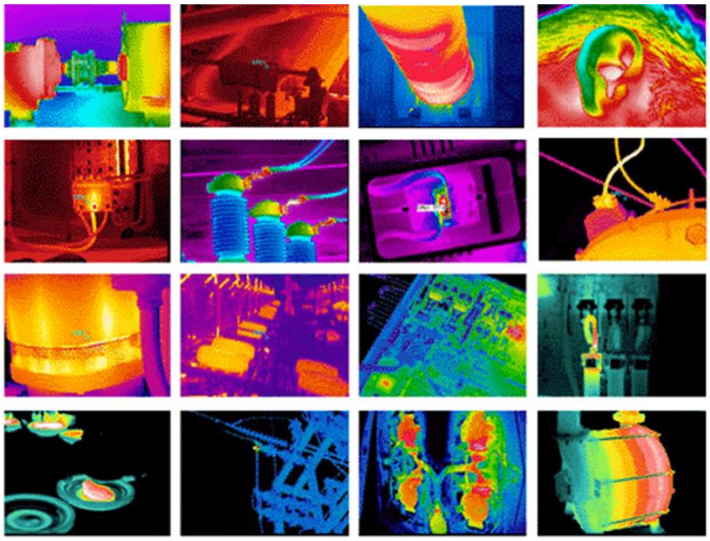

Ilustración 4. Imágenes termográficas (Serrano Malagón & Núñez Campo, 2011)

2.3 TIPOS DE SENSORES DE TEMPERATURA

Como se ha descrito en el apartado anterior, a lo largo de la historia han ido surgiendo distintos métodos para medir la temperatura. La consecuencia de esto es que en la actualidad haya una amplia variedad de sensores de temperatura. En este apartado se van a analizar los tres tipos más empleados: sensores RTD, termopares y sensores infrarrojos(*Tipos de SENSORES de Temperatura y Sus Diferencias*, n.d.).

2.3.1 TERMOPARES

Suelen estar compuestos por dos hilos metálicos unidos por un extremo, llamado junta de medición o junta caliente. El extremo opuesto, se llama junta fría. Su funcionamiento consiste en producir una tensión que es proporcional a la temperatura del sistema ante una variación de temperatura. En el mercado hay infinidad de termopares, pero los más comunes son los de tipo J, K y T. El que un termopar pertenezca a una familia u otra dependerá del material del que esté compuesto.

- Termopar tipo J: Está compuesto de hierro y constatan (aleación de cobre y níquel). Tiene su uso limitado en entornos que sean oxidantes. Su rango de temperatura va de 0ºC a 750ºC.
- Termopar tipo K: Se compone de una junta de chromega (aleación de cromo y níquel) y alomega (aleación de aluminio y níquel). Este termopar es el más usado. Se pueden medir temperaturas desde los -200ºC hasta los 1250ºC.

- Termopar tipo T: Hecho de una combinación de cobre y constatan. Se recomienda en entornos húmedos. Cuenta con un rango de temperatura de -250ºC hasta los 350ºC.

Por lo tanto, a la hora de elegir uno u otro termopar deberemos tener en cuenta los siguientes parámetros:

- Donde se va a emplear el sensor.
- Que rango de temperatura es el óptimo.
- Resistencia que tiene a la vibración y a la abrasión.
- Resistencia química del termopar.

Ilustración 5. Ejemplo de termopar(Lizán Ortiz et al., 2018)

2.3.2 SENSORES INFRARROJOS

Miden la radiación térmica y electromagnética en el espectro infrarrojo. En este espectro, las longitudes de onda van desde los 0,7μm hasta los 1000μm. Una manera de clasificarlos es atendiendo a las longitudes de onda que son capaces de medir:

- Infrarrojo cercano: Se pueden medir longitudes de onda desde los 700nm a los 3μm.
- Infrarrojo medio: Miden longitudes de onda entre 3μm hasta 60μm.
- Infrarrojo lejano: Se emplean con longitudes de onda desde 60μm hasta 1000 μm.

Este tipo de sensor se emplea cuando la medida se quiere realizar son contacto, ya sea porque no se puede acceder al objeto del cual se quiere obtener la temperatura o porque resulte más cómodo. Una analogía de este tipo de sensor, son las cámaras de visión térmica.

Ilustración 6. Ejemplo de sensor infrarrojo

2.3.3 SENSORES TERMORRESISTENTES

Una de las principales características que presentan estos sensores, es su elevada fiabilidad a la hora de medir temperaturas muy elevadas. Para ello, suelen estar hechos con materiales compuestos, cerámicas o metales especiales. Pueden llegar a soportar temperaturas por encima de los 100ºC. Los principios que emplean para medir la temperatura varían en función del tipo:

- RTD: Emplean el platino como elemento sensible. Son muy estables y precisos cuando trabajan a elevadas temperaturas.
- Termistores de cerámica: Es muy común emplear cerámica de óxido de aluminio. Son muy empleados cuando se necesita medir una alta gama de temperaturas.
- Sensores de fibra óptica: Emplean la fibra óptica como medio por donde se transmiten y miden los cambios de temperatura.

Ilustración 7. Ejemplo de sensor de fibra óptica (Cámara, 2017)

2.3.4 CÁMARAS TÉRMICAS

Las cámaras térmicas capturan la radiación infrarroja de objetos y a partir de ella generan imágenes. Gracias a ello muestran imágenes que muestran variaciones de temperatura.

Las características de este tipo de sensor son las siguientes:

- Funcionan en el espectro infrarrojo detectando el calor que emiten los objetos.
- Detectan variaciones muy pequeñas de temperatura, lo que las hace un sensor muy sensible.
- Son útiles en condiciones ambientales difíciles, ya que pueden ver a través de polvo, humo, niebla…
- Aplicaciones: Seguridad y vigilancia, inspecciones industriales, imágenes médicas…

2.4 ESTUDIO DE MERCADO

2.4.1 TERMOPAR TIPO K

Se podría construir el sistema de control de temperatura a partir de un termopar tipo K. Todos los componentes del sistema se pueden obtener del fabricante SparkFun y serían los siguientes:

- Termopar Tipo K: Bastaría con seleccionar uno de los termopares de este tipo que ofrece el fabricante.
- Conector del Termopar: Su función es la de conectar el termopar al módulo.
- Módulo MAX31855K: Como el termopar proporciona una salida analógica, este módulo es necesario para convertirlas a digitales y así poder leerlas.
- Microcontrolador: Para procesar las lecturas. Por ejemplo, un microcontrolador ESP32.
- Interfaz de Visualización: Necesaria para poder ver los resultados de las temperaturas en tiempo real.
- Comunicación: Se necesita un módulo para establecer la comunicación con el ordenador. Por ejemplo, SparkFun FTDI Basic Breakout - 3.3V

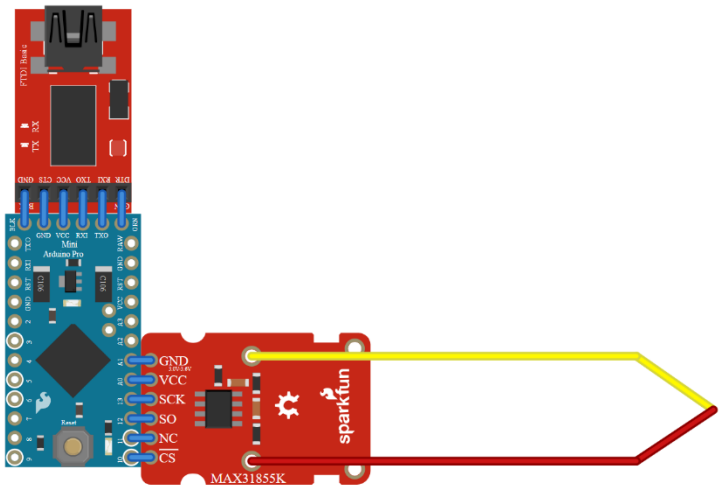

Ilustración 8. Circuito del termopar similar al propuesto (MAX31855K Thermocouple Breakout Hookup Guide - SparkFun Learn, n.d.)

2.4.2 TERMISTOR TMP36

Otra opción sería emplear el termistor TMP36, nuevamente usando elementos del fabricante mencionado en la sección 2.4.1. Los componentes para construir el sistema serían:

- Termistor TMP36: Se trata de un sensor de temperatura analógico que proporciona una salida de voltaje proporcional a la temperatura.
- Microcontrolador: Al igual que en el caso anterior, escogemos por ejemplo el ESP32.
- Interfaz de visualización: Para este circuito se escoge una pantalla LED.
- Potenciómetro: Se va a usar para regular el contraste de la pantalla y que así se vean de manera correcta las lecturas de temperatura.

Como apunte, observar que para este tipo de sensor no es necesario incorporar el módulo para transformar las lecturas de temperatura ya que la lectura, como se ha indicado, es proporcional a la temperatura.

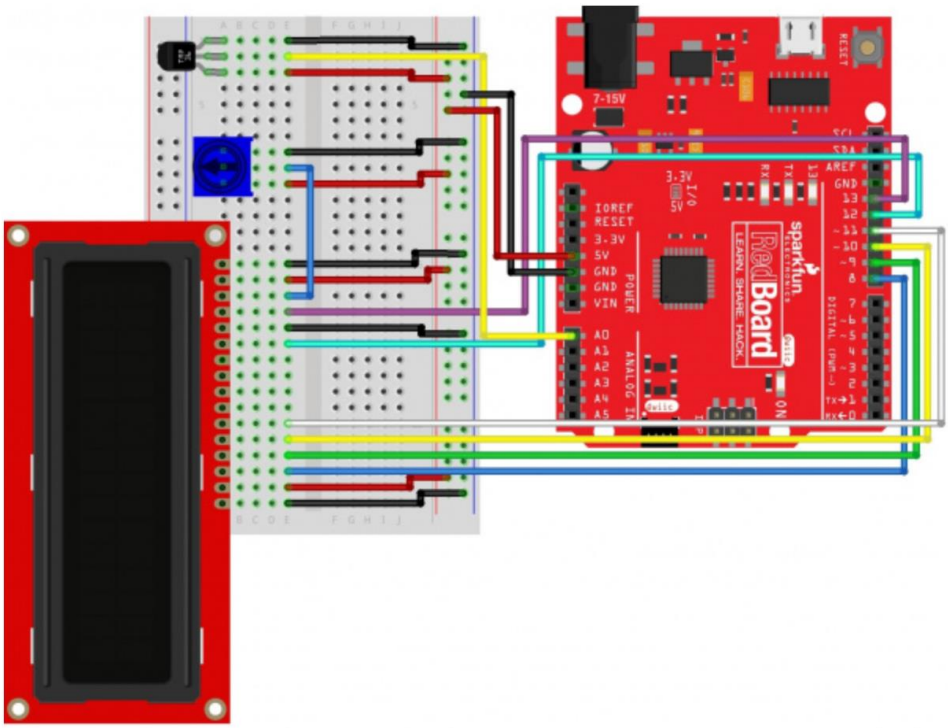

Ilustración 9. Ejemplo de circuito para sensor TMP36 (SparkFun Inventor's Kit Experiment Guide - v4.1 - SparkFun Learn, n.d.)

2.4.3 CÁMARA TÉRMICA M5STICK T-LITE

A diferencia del resto de los sensores, este dispositivo nos ofrece el sistema de control de temperatura ya montado. Incorpora un microcontrolador ESP32 y el sensor de imágenes infrarrojas MLX90640; así como una pantalla a color que muestra información en tiempo real.

Además, posee capacidades de conexión Wi-Fi y Bluetooth integradas lo que permitirá la conexión inalámbrica con otros dispositivos o sistemas. También ofrece una interfaz API de datos en línea, que puede obtener imágenes remotas y los datos correspondientes a las mismas a través de EZData.

Ilustración 10. M5Stick T-Lite (T-Lite, n.d.)

2.4.4 CONCLUSIÓN

Tras comparar las distintas opciones con que se puede realizar el sistema de control de temperatura, se decide que la mejor opción por sencillez, operabilidad y prestaciones es la cámara térmica M5Stick T-Lite.

Este dispositivo proporciona mediciones precisas y en tiempo real de la temperatura, además de tener la capacidad de capturar imágenes térmicas en tiempo real y poder ver los patrones de temperatura. Otras de las ventajas que ofrece, es la capacidad de transmitir datos de manera remota a través de su interfaz API de datos en línea.

Además, también se tiene la posibilidad de incorporar módulos externos para emplear LoRa y así poder comparar esta comunicación con la comunicación vía Wi-Fi que ya incorpora el propio sensor.

2.5 REDES NEURONALES

2.5.1 DEFINICIÓN DE RED NEURONAL

Las redes neuronales artificiales se tratan de un sistema computacional que se asemeja en gran medida a la manera de procesar información del cerebro humano. Es decir, sigue el mismo principio que el cerebro humano, almacenamiento de información distribuido y procesamiento colaborativo de dicha información. Es un sistema dinámico que se basa en la conexión de neuronas a las que se les asigna un trabajo sencillo. Como estas neuronas pueden trabajar de manera paralela, se puede llegar a tener una gran velocidad de procesamiento de datos (Cui & Wu, 2015).

Las aplicaciones actuales de las redes neuronales son entre otras: robótica, medicina, defensa, coches, industria aeroespacial, etc. Tienen un papel muy importante en la predicción, monitorización, control del estado y diagnóstico de la maquinaria y equipos de la industria.

2.5.2 ENTRENAMIENTO DE LA RED

La estructura básica de una red neuronal consiste en una capa de entrada, una capa oculta y una capa de salida. Así, en el diagnóstico, el vector de la capa de entrada es aquel en el que se almacenan los síntomas de fallo de la maquinaria y cada neurona en dicho vector son distintos fallos. En el vector de salida se almacenan las categorías de cada fallo y cada neurona corresponde a un tipo de fallo. Si por ejemplo hay cinco tipos de fallos,

habrá cinco neuronas en la capa de salida. En la capa oculta, se utilizan tantas neuronas como se estimen oportunas.

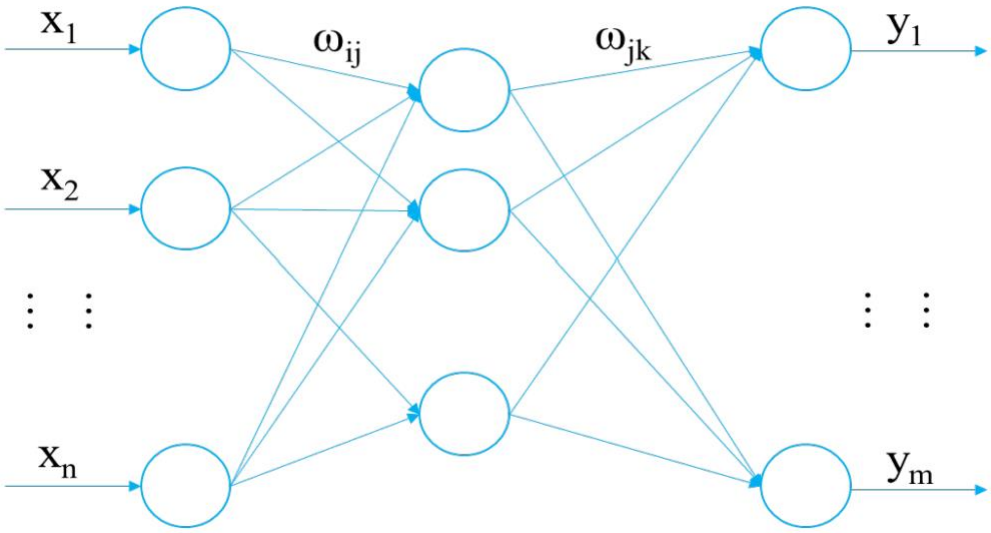

Ilustración 11. Esquema básico de una red neuronal (Du, 2023)

El proceso de cálculo de la red neuronal es de la siguiente manera:

1. La información entra en cada neurona de la capa de entrada.

2. A las conexiones entre neuronas se les asigna un peso que está almacenado en las neuronas de la capa oculta en función de que dos neuronas estén conectadas.

3. La información de la neurona de entrada se multiplica por el valor de esa conexión y se le suma el valor de la neurona de la capa de salida.

2.5.3 FUNCIONES DE ACTIVACIÓN

Las funciones de activación son funciones que se emplean en las capas ocultas de las redes neuronales, para que las neuronas manejen de una forma óptima los diferentes datos que debe procesar. La manera en que trabajan sería la siguiente: en una neurona entra información de la capa anterior, se le suma el peso de la neurona en que se encuentra y por último se le aplica la función de activación.

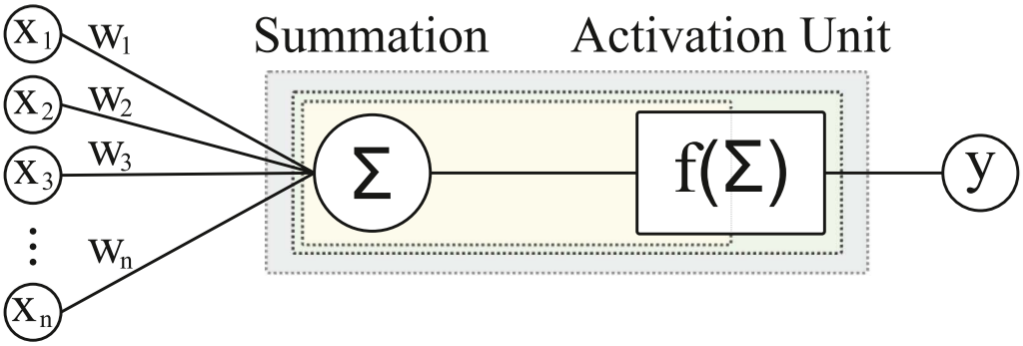

Ilustración 12. Red neuronal con función de activación (Pappas et al., 2023).

Hay muchos tipos, pero se puede considerar como funciones clásicas a emplear las siguientes: *step*, *sigmoid* y *tanh*. El objetivo de estas funciones es, durante el entrenamiento de la red, transformar gradientes con un elevado valor en gradientes cercanos a cero. Esto constituye un problema en redes cuyo objetivo sea clasificar o reconocer distintos patrones, ya que no hay cambios significativos en los pesos de las neuronas (Amin et al., 2023).

Es por ello por lo que surgen nuevas funciones de activación como *ReLu*, *Softmax*, *SoftPlus*, *Parametric ReLu*, etc.

2.5.4 REDES NEURONALES CONVOLUCIONALES

Las redes neuronales convolucionales se inspiran en la corteza visual del cerebro e introducen dos tipos de capas nuevas: las capas de convolución y las capas de agrupación.

Este tipo de redes han ganado importancia en el campo de las inteligencias artificiales, sobre todo en el reconocimiento de patrones y en el procesamiento de imágenes. La mayor diferencia que presentan respecto de las redes tradicionales es su capacidad para procesar patrones espaciales y la correlación de dados bidimensionales (Dai, 2021)

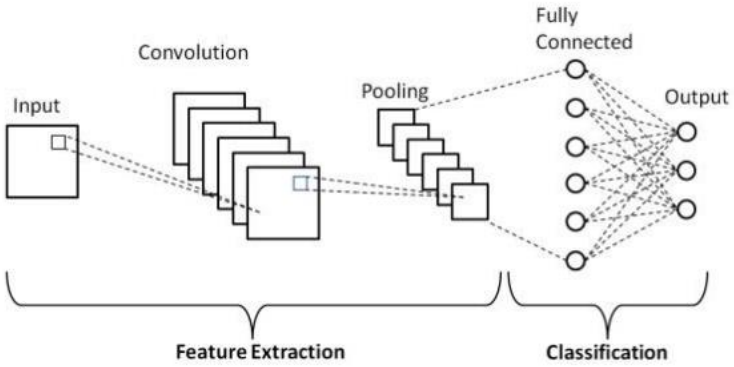

Ilustración 13. Estructura de una CNN(Dai, 2021)

La manera en que trabaja una capa de convolución es aplicando filtros a la imagen, de tal manera que se resaltan ciertos patrones de esta como los bordes o ciertas texturas. En cuanto a las capas de agrupación, se aplican después de las capas de convolución para eliminar aquellas partes que no son representativas de la imagen. Es decir, disminuyen el número de pixeles de la imagen para que su procesamiento posterior sea más sencillo.

Si se juntan estos dos tipos de capas (y se pueden apilar tantas como se quieran, es decir: convolución – agrupación – convolución – agrupación …) una red trabajaría de la siguiente manera:

1. La imagen está dividida en pixeles y cada píxel tiene un valor numérico en función del color de este. Por lo tanto, necesitaremos en la capa de entrada tantas neuronas como pixeles tenga nuestra imagen.

2. Los pixeles son evaluados junto con los contiguos para detectar si hay cambios significativos en el valor numérico y así detectar cambios de patrones. Esto se hace aplicando núcleos a ese conjunto de pixeles que se desean evaluar.

3. Una vez que hemos aplicado ese filtro(núcleo) a nuestra imagen y ya hemos variado el valor de los pixeles para detectar cambios significativos, la imagen entra a la capa de agrupación.

4. La capa de agrupación toma los pixeles por ejemplo en matrices de 3x3, y de esa matriz solo se queda con el píxel de mayor valor.

5. Este proceso se puede repetir tantas veces como se quiera, para posteriormente pasar esos datos por las capas ocultas.

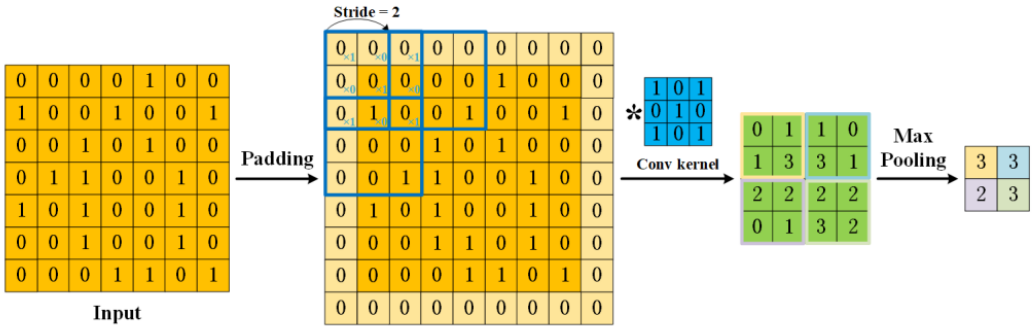

Ilustración 14. Funcionamiento de las capas de una CNN (Li et al., 2022)

Un ejemplo práctico en donde se emplearon este tipo de redes fue durante el COVID-19. Mediante una red neuronal de este tipo se identificaban en radiografías de tórax si el paciente presentaba o no afección respiratoria (Dandil & Yildirim, 2022).

2.5.5 REDES NEURONALES RECURRENTES

Este es un tipo de red neuronal que nos va a permitir procesar datos de tipo secuencial, en el que el orden de estos es bastante importante. Por ejemplo, se usa en las predicciones meteorológicas al ser importante como han sido las condiciones meteorológicas en días anteriores. Pero en donde más se emplea es para el procesamiento de audio o de texto ya que es muy importante la manera en que se dicen las oraciones. Así, es el tipo de red que se emplea en los sistemas de reconocimiento por voz.

La mayor diferencia que presentan respecto a los otros tipos de redes hasta ahora vistas es que esas son las óptimas a la hora de clasificar datos y ver patrones, pero no se pueden emplear para generar datos al utilizar solo el dato de entrada de ese momento sin tener en cuenta los datos previos. Esto se consigue haciendo un bucle en la red neuronal como se muestra en la imagen inferior (Kovacs et al., 2021)

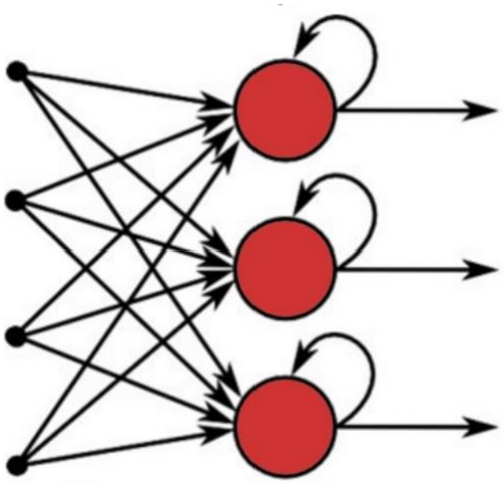

Ilustración 15. Estructura básica de una RNN (Kovacs et al., 2021)

Como se puede ver en la imagen, cada neurona tiene dos entradas y dos salidas. En este tipo de red cada carácter o cada palabra de una secuencia que compone una salida en cada paso se denomina instante de tiempo. Por ejemplo, si la salida en un determinado momento es la letra "i" ese sería el instante de tiempo n. La otra salida de la red es el estado oculto correspondiente a ese estado de tiempo. Cada estado oculto es una función que depende de todos los estados ocultos anteriores. Las dos entradas son el estado de tiempo anterior y el estado oculto anterior.

Una aplicación interesante para este tipo de redes es la predicción del valor de ciertas acciones en el mercado de valores. Por ejemplo, en (Kovacs et al., 2021)se empleó una base de datos de los históricos del NASDAQ para intentar adivinar las tendencias futuras de las acciones.

Uno de los problemas principales que presentan es que tienen una memoria de corto plazo. Esto ocurre porque, como ya se ha dicho, el estado oculto depende de todos los estados ocultos anteriores pero la información se va dando de forma anidada. Así, el último estado oculto tendrá mucho más peso que el estado oculto inicial. Este problema se soluciona con las redes LSTM. Con esta arquitectura se consigue almacenar los datos que son relevantes para la red y se desechan los datos que no son ya necesarios. Es decir, tienen la capacidad de añadir o eliminar información en función de la importancia de esta. Lo que hacen es incorporar una celda de estado al instante de tiempo y funciona como la memoria. Internamente es como si añadiésemos una red neuronal que hace la función de distinguir entre información relevante e información irrelevante (Gill et al., 2023).

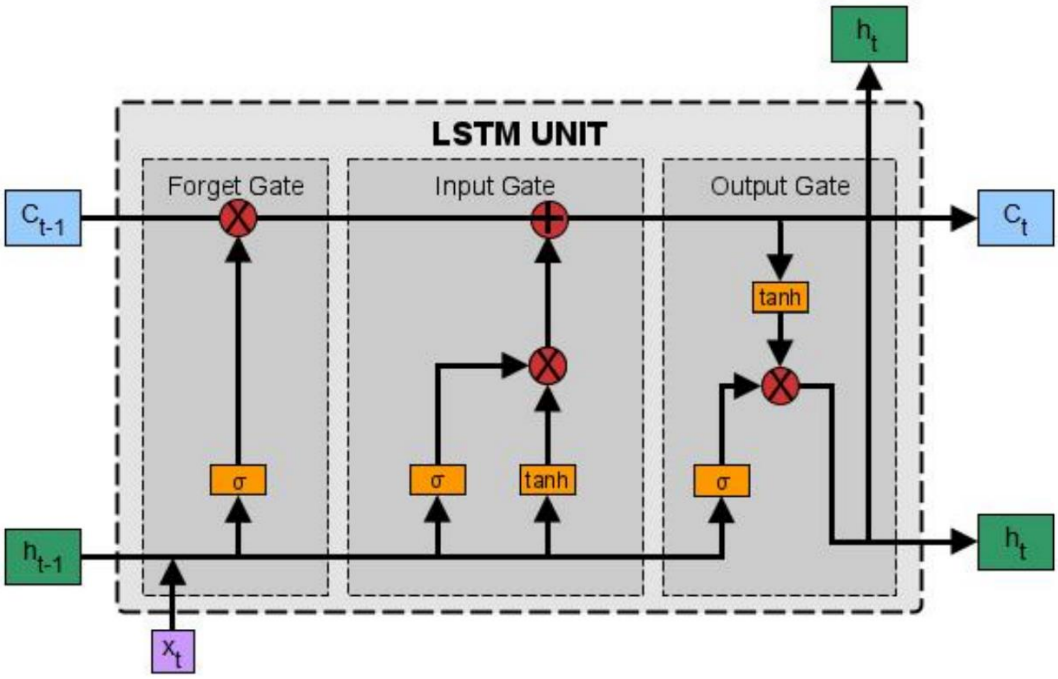

Ilustración 16. Arquitectura de una red LSTM (Kovacs et al., 2021)

Las distintas arquitecturas que presentan estas redes son:

- *One to many*: Tienen una única entrada y la salida es una secuencia. Por ejemplo, se pueden emplear para describir una imagen; la imagen sería la entrada y la descripción proporcionada la secuencia.
- *Many to one*: Son la red inversa a la anterior. Se puede emplear para clasificar textos.
- *Many to many*: La entrada es una secuencia y la salida es otra secuencia. Son muy empleadas en traductores de texto.

2.5.6 REDES GENERATIVAS ADVERSARIAS

Este tipo de red está compuesta de otras dos redes neuronales que son enfrentadas entre ellas. Por un lado, está la red generadora y por otro lado la red discriminadora. Cada una de estas redes cumple una función dentro de la GAN y ha quedado demostrado la gran utilidad de estas. Principalmente se utiliza con imágenes, pero también se pueden emplear con audio y video (Prabhat et al., 2020).

La red generadora se encarga de crear datos sintéticos que tratan de replicar datos reales. Para llevar a cabo este proceso, tiene como entrada un vector de números aleatorios al que se conoce como ruido.

La red discriminadora trata de diferenciar los datos reales de los generados por la otra red. De esta manera, a medida que la red va siendo entrenada va a ser capaz de diferenciar de manera más exacta los datos reales de los datos falsos (Cobelli et al., 2022)

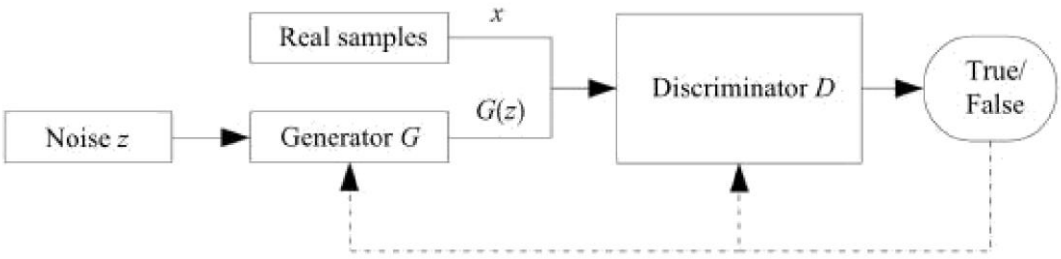

Ilustración 17. Diagrama de bloques de una GAN (Kumar & Dhawan, 2020)

Teniendo en cuenta la estructura de este tipo de redes, la manera de entrenarla y funcionamiento interno sería el siguiente:

1. Se introducen datos reales de un *dataset* a la red que hace de discriminador, así como datos falsos que deberán ser del mismo tamaño que los que genere el generador en el modelo. Es así como la red comienza a entrenarse y a diferenciar datos reales de datos falsos. El discriminador etiqueta los datos y estos van a servir como entrada a la propia red.

2. A continuación, se comienza a introducir el ruido, a partir de una distribución Gaussiana aleatoria, en la red generadora para que comience a generar datos que van a ser posteriormente introducidos a la red discriminadora y que esta los clasifique en verdaderos o falsos. De esta manera, el generador va a ir generando datos cada vez más reales.

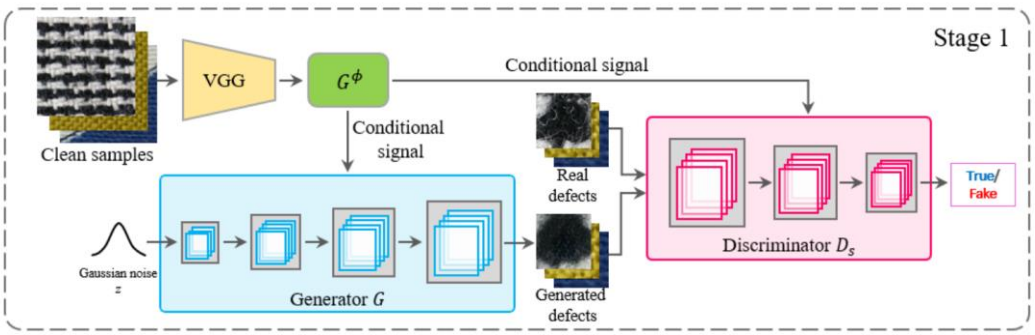

Ilustración 18. Esquema del entrenamiento de una GAN (Liu et al., 2020)

Dentro de las redes tipo GAN existen distintas variantes (Karthika & Durgadevi, 2021):

- *VGAN*: Son redes en las que el discriminador y el generador son multicapa, pero siguen el mismo esquema que las GAN normales.
- *FCGAN*: Se emplean con imágenes sencillas empleando redes neuronales totalmente conectadas.
- *LAPGAN*: Se emplea más de un generador o de un discriminador.
- *CGAN*: La red está condicionada con algunos datos externos auxiliares. Suele tener la misma estructura que una red VGAN.
- *DCGAN*: Son las más empleadas. En lugar de ser multicapas, emplean redes neuronales convolucionales.

Este tipo de redes son ampliamente usadas para generar nuevas imágenes para *datasets* y así poder aumentar el número de entradas de entrenamiento para otro tipo de redes.

Capítulo 3. DEFINICIÓN

3.1 MOTIVACIÓN

El porqué del presente trabajo surge como una necesidad dentro de las industrias del siglo XXI. Desglosando el título del proyecto, se puede llegar a tres motivos que dan pie a realizarlo.

En primer lugar, y como ya se indicó, los sistemas de control de temperatura conforman un pilar fundamental en los procesos productivos. En la mayor parte de las industrias es incluso el parámetro más importante, ya que con su control y regulación se obtendrá un producto de calidad con una buena eficiencia del proceso. Además de motivos puramente económicos también es esencial para asegurar que se está trabajando en un ámbito seguro y así reducir el número de accidentes laborales.

El segundo motivo que impulsa este trabajo es la Industria 4.0. En un ámbito de constante digitalización y automatización, se hace indispensable la integración de los sistemas de control en la misma para poder tomar decisiones en tiempo real.

Por último, el tratar de implementar todo lo dicho anteriormente con sistemas de baja potencia. La integración de este tipo de tecnología permite que la eficiencia energética mejore, a la par que aumenta la conectividad tanto en distancia como en número de dispositivos. Por tanto, llegamos a un sistema que es más eficiente energéticamente hablando, que nos aporta una mayor conectividad y además es fácilmente escalable.

Además, también surge de un interés personal de aprender de un ámbito del que hasta hace poco uno mismo no tenía mucha información y que desconocía por completo; como son las citadas redes de baja potencia, así como un código de programación que no sabía emplear antes del inicio de este proyecto y que es muy útil en la actualidad.

3.2 OBJETIVOS

A continuación, se enumeran los cinco objetivos principales que se persiguen con este Trabajo de Fin de Máster:

1. **Diseñar y configurar el sistema:** Crear un sistema que se emplee en monitorear y controlar la temperatura en distintos tipos de procesos. Para ello se va a utilizar la cámara termográfica M5stick T-Lite.

2. **Crear un código:** Una de las fases fundamentales de este proyecto, consiste en generar el código. Posteriormente será cargado en el microcontrolador ESP32 que está integrado en la cámara térmica. Este objetivo es muy importante, pues es el que permitirá la correcta operación y comunicación del dispositivo.

3. **Establecer límites:** se procederá a la definición de límites térmicos críticos en función de los requerimientos específicos de cada proceso. Estos límites actuarán como umbrales de alarma, generando alertas en caso de superarse, lo que garantiza una supervisión precisa y oportuna.

4. **Comunicación con interfaz:** Se pretende poder monitorear las temperaturas que se vayan obteniendo, para así poder realizar el control de estas.

5. **Aplicación a un proceso:** Se probará el sistema en procesos simulados y en procesos reales para demostrar su funcionalidad.

En resumen, se quiere diseñar un sistema funcional para poder controlar la temperatura de un proceso y para ello se empleará una cámara termográfica. Se pasarán por distintas fases hasta conseguir probar la misma en un ámbito real, pudiendo demostrar que el sistema funciona en la manera en que se pretende y demostrando que es posible su implantación en un proceso productivo de la Industria 4.0.

3.3 METODOLOGÍA

En este capítulo se presenta como se pretende realizar el proyecto, haciendo una estimación temporal del mismo:

JUNIO-JULIO 2023

- Búsqueda bibliográfica sobre los sistemas de control de temperatura, los sistemas de baja potencia y la industria 4.0.
- Definición de objetivos y metodología a seguir durante el proyecto.

AGOSTO 2023

- Estudio de mercado sobre los distintos sistemas de temperatura.
- Decisión del sistema de control de temperatura a emplear.

SEPTIEMBRE-OCTUBRE 2023

- Comienzo de la redacción de la memoria del TFM: Introducción, Estado del Arte, Estado de la Cuestión.

- Decisión de complementos a emplear junto con el sistema de control de temperatura.

NOVIEMBRE 2023

- Encargo y recepción del hardware a emplear en el proyecto.

DICIEMBRE-ENERO 2023

- Programación del sistema
- Primeras pruebas con el sistema Wi-Fi.

FEBRERO-MARZO 2024

- Refinamiento del código.
- Pruebas con el sistema LoRa y comparación con el Wi-Fi.

ABRIL 2024

- Conclusiones del trabajo y trabajos futuros.
- Finalización de la memoria.

3.4 PLANIFICACIÓN

A continuación, se indica la planificación real del trabajo siendo como era de esperar distinta a la estimada en el apartado anterior:

ACTIVIDAD	FECHA DE INICIO	FECHA DE FINALIZACIÓN	DÍAS
1. BÚSQUEDA DE INFORMACIÓN	01/06/2023	17/03/2024	290
Búsqueda de información general	01/06/2023	31/07/2023	60
Estudio de mercado	01/08/2023	31/08/2023	30
Elección del sistema a emplear	01/08/2023	31/08/2023	30
Lectura sobre redes neuronales	10/01/2024	17/03/2024	67
2. REDACCIÓN DE LA MEMORIA	01/09/2023	16/06/2024	289
Introducción	01/09/2023	02/09/2023	1
Estado del arte	01/09/2023	17/03/2024	198
Definición de objetivos y metodología	01/06/2023	31/07/2023	60
Desarrollo	18/03/2024	09/06/2024	83
Conclusión	10/06/2024	11/06/2024	1
Líneas Futuras	12/06/2024	13/06/2024	1
Anexos	18/03/2024	16/06/2024	90
3. PROGRAMACIÓN DEL SISTEMA	22/01/2024	15/05/2024	114
Descarga e instalación de sistemas necesarios	22/01/2024	28/01/2024	6
Pruebas con sistemas más sencillos	30/01/2024	18/02/2024	19
Programación del sistema sin A.D	19/02/2024	24/03/2024	34
Programación del sistema con A.D	25/03/2024	27/04/2024	33
Refinación del código	28/04/2024	15/05/2024	17

Tabla 1. Planificación del proyecto

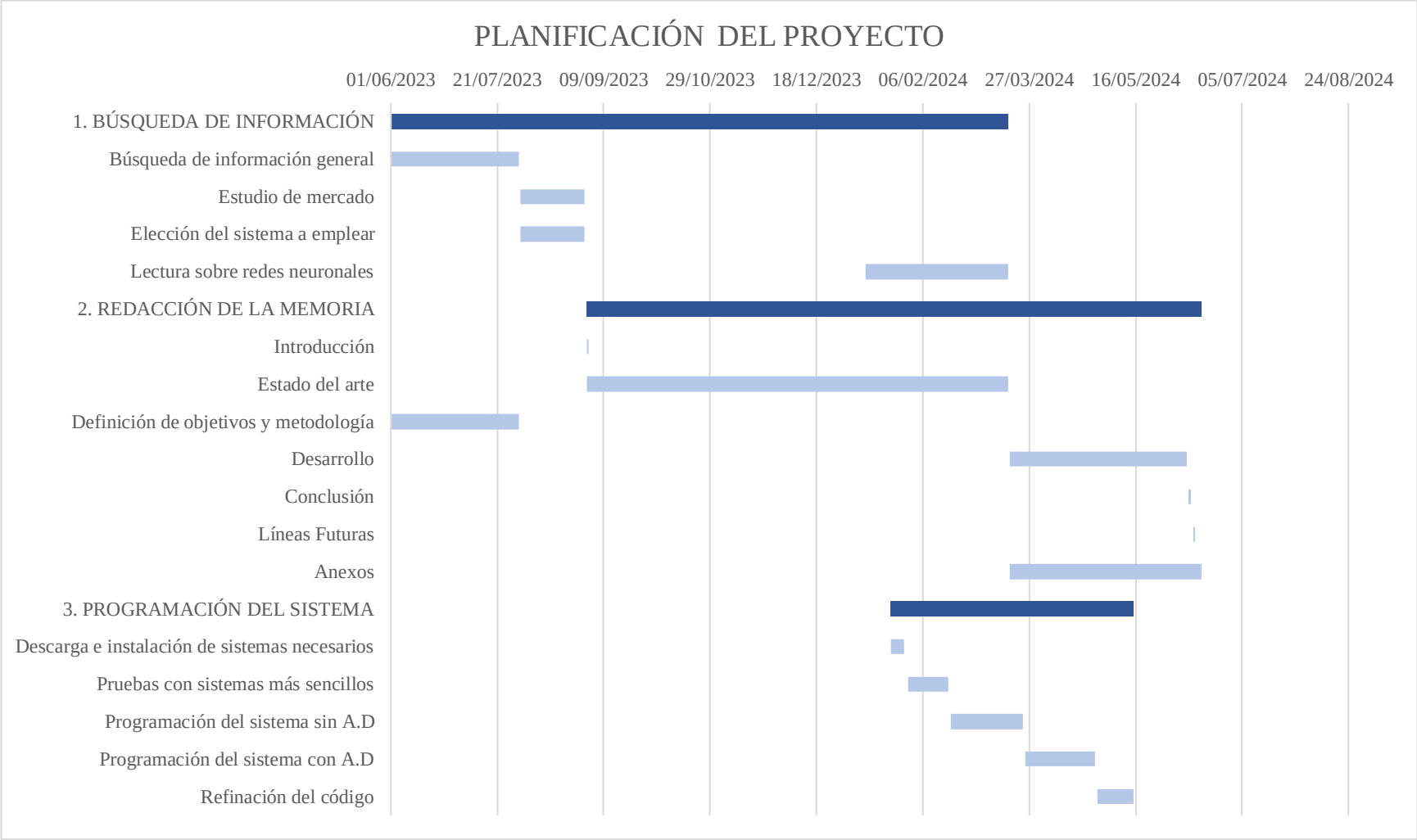

Tabla 2. Diagrama de Gantt

Capítulo 4. DESARROLLO

4.1 TIPO DE RED SELECCIONADA

De las tres redes expuestas en el Capítulo 2 (y otras muchas que no han sido mencionadas) la óptima para el análisis de imágenes térmicas de motores son las redes neuronales convolucionales. Es por ello por lo que se decide construir una red de este tipo en el presente trabajo. Las razones que han motivado esta decisión son las siguientes:

1. Las redes neuronales convolucionales tienen como fin el procesamiento de imágenes, por lo que es la opción a priori más lógica para el análisis de las imágenes que se van a emplear. Las capas de convolución anteriormente explicadas van a permitir identificar distintos patrones en los datos aportados pudiendo clasificarlos de manera exacta.

2. Las capas de agrupación permiten reducir la dimensión de las imágenes, lo cual va a permitir realizar un mejor procesamiento de estas. Esto es muy beneficioso para el caso en que se trabaje con imágenes que tengan una alta resolución (como es el caso de las imágenes térmicas).

3. Las redes generativas adversarias, podrían ser una buena herramienta en un proyecto futuro en el que se necesite generar imágenes sintéticas para así aumentar la base de datos y poder mejorar el entrenamiento de la red. Sin embargo, para esté punto de partida del proyecto no son la mejor opción ya que su aplicación sería menos directa al poder generar imágenes que no fuesen perfectas y por lo tanto distorsionando los resultados, algo no deseable en esta etapa inicial del proyecto.

4. En cuanto a las redes neuronales recurrentes son una fantástica herramienta para el procesamiento de datos secuenciales, pero no para el análisis y clasificación de imágenes. Es evidente que en el proyecto que se pretende desarrollar, las imágenes térmicas no presentan una estructura de datos secuenciales y por lo tanto este tipo de redes queda totalmente descartado.

5. Otra de las ventajas que presentan las CNN es su adaptabilidad a un amplio abanico de problemas. Al poderse modificar los hiperparámetros y la arquitectura, se puede optimizar el modelo y prepararlo de la mejor manera para tratar las imágenes térmicas. Gracias a esta característica se va a poder optimizar la red para llegar a un mejor resultado en el análisis.

6. Las redes neuronales convolucionales pueden ser computacionalmente muy intensas durante el entrenamiento, pero, una vez que se ha entrenado el modelo son eficientes en cuanto al tiempo para detectar errores en las imágenes térmicas. Esto permite una inferencia rápida en tiempo real, lo cual es esencial para aplicaciones del mundo real que requieren una respuesta rápida para detectar y diagnosticar fallos del motor.

En conclusión, la selección de este tipo de red se realiza por que están especializadas en el análisis y el procesamiento de imágenes, las capas de agrupación permiten reducir el tamaño de las imágenes, la adaptabilidad a diversos tipos de problemas y la capacidad de análisis que proporcionan. Las redes generativas adversarias no quedan del todo descartadas ya que podrían ser muy útiles en estados más avanzados de este proyecto o incluso en proyectos futuros. Por último, las redes neuronales recurrentes quedan completamente descartadas ya que están especializadas en el tratamiento de datos secuenciales.

4.2 HERRAMIENTAS EMPLEADAS

4.2.1 LENGUAJE DE PROGRAMACIÓN

Para el desarrollo de una red neuronal se puede emplear prácticamente cualquier lenguaje de programación. No obstante, hay algunos que proporcionan una flexibilidad y unas herramientas de las que carecen otros. Los dos lenguajes más empleados suelen ser C++ y Python.

C++ surge como una extensión del lenguaje de programación C con el fin de incorporar herramientas que permitiesen manipular objetos. Presenta un alto rendimiento ya que posee una gran cantidad de parámetros para la optimización y está en constante actualización.

No obstante, presenta una gran desventaja como es la complejidad a la hora de resolver los problemas que surgen de los errores de escritura del código. También se complica a la hora de manejar las librerías.

Las desventajas que presenta C++ llevan a realizar el presente trabajo en el lenguaje de programación Python. Este lenguaje también presenta alguna desventaja como puede ser la velocidad de ejecución menor que presenta con respecto a otros o los problemas que pueden surgir para el desarrollo móvil. Sin embargo, estos inconvenientes no representan un problema para el desarrollo de la red neuronal que se pretende diseñar.

Python presenta muchas ventajas para el desarrollo de este tipo de redes, así como para personas que no estén familiarizadas con ellas. Esto se debe a la sencillez que presenta su sintaxis, la cual no va a requerir una gran cantidad de tiempo para el desarrollo y revisión del código. A su vez, posee un gran número de librerías que están especializadas en el aprendizaje automático como pueden ser *TensorFlow, Scikit-Learn, PyTorch* o *Keras*. Estas bibliotecas van a facilitar en gran medida el desarrollo y entrenamiento de la red. Otra de las ventajas que presenta es la integración con otros lenguajes de programación, lo cual puede ser de gran ayuda en el caso de tener que emplear algún sistema ya existente.

Además de las ventajas mencionadas, Python también destaca por la adaptabilidad una gran variedad de situaciones en el ámbito del aprendizaje automático y como se trata de una herramienta de código abierto se torna muy sencillo encontrar información en este lenguaje de programación.

4.2.2 TENSORFLOW

Esta herramienta es una librería ampliamente usada para el desarrollo de sistemas. Se trata de una plataforma muy popular en el aprendizaje automático de redes (caso de una red neuronal en Python). *TensorFlow* está desarrollado por *Google Brain* basándose en redes neuronales de aprendizaje profundo y ofrece una escalabilidad y flexibilidad que va a facilitar mucho el entrenamiento y construcción de la red neuronal. Además, incluye una gran cantidad de bibliotecas y herramientas que son complementarias. Una de las más interesantes es *TensorBoard* la cual sirve para visualizar el proceso de ejecución de *TensorFlow*, así como distintas métricas que resultarán de gran ayuda.

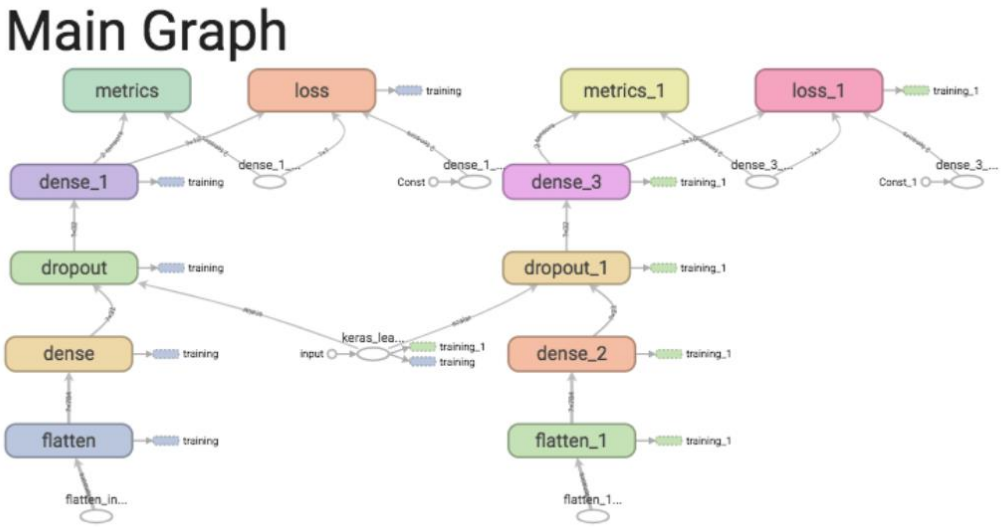

Ilustración 19. Ejemplo de un grafo en TensorBoard (Web Oficial de TensorFlow, n.d.)

Esta librería de código abierto está generada en C++ y en Python pudiéndose correr en múltiples CPUs y GPUs. Es multiplataforma: Windows, Mac, Linux 64, Android e IOs; y proporciona una Api muy documentada en Python y menos en C++, Java y Go. En resumen, se trata de la herramienta más popular de *Machine Learning* sobre todo por su efectividad y potencialidad.

4.2.3 KERAS

Al igual que *TensorFlow* es una biblioteca de código abierto, pero en este caso solo está escrita en Python y desarrollada por Google. Se comporta como una API ya que su objetivo es el de acelerar y facilitar la creación de redes neuronales. Con esta herramienta se pueden crear los modelos de manera eficiente y muy rápida gracias a su sintaxis intuitiva y muy sencilla al no ser códigos muy complejos. Así, nos ofrecerá todo lo necesario para la configuración de las capas necesarias en la red y las funciones de activación a emplear.

Por lo tanto, *TensorFlow* nos facilita la manera en que van a ser ejecutados los modelos y Keras nos proporciona las herramientas necesarias para el proceso de creación y entrenamiento de la red neuronal.

4.2.4 OTRAS HERRAMIENTAS

Dentro del código que se va a desarrollar también se van a emplear las siguientes herramientas:

1. *ImageDataGenerator*: Se emplea para realizar el aumento de datos de las imágenes.

2. *Os*: Es un módulo de Python que se va a usar para leer las rutas donde se encuentran las imágenes del dataset.

3. *Numpy*: Es la librería que se va a emplear para los arreglos multidimensionales. Se empleará en la validación de los datos.

4. *Matplotlib*: Se emplea para imprimir por pantalla las gráficas de *accuracy* y de pérdida. También se puede emplear para mostrar alguna imagen del dataset.

4.3 DATASET

El conjunto de datos que se va a emplear son un conjunto de imágenes térmicas obtenidas con dos resoluciones distintas:

- Imágenes de 640x512 pixeles que han sido tomadas con la cámara *Workswell WIC* 640.
- Imágenes de 160x120 pixeles obtenidas con la cámara *Fluir Lepton* 3.5

Además, se han clasificado esas imágenes en tres estados distintos del motor:

1. Motor funcionando correctamente: El motor está funcionando de manera correcta en condiciones normales de operación.

2. Motor desalineado: El motor y el instrumento conectado al mismo no están alineados de manera correcta debido al desgaste de alguna pieza.

3. Motor estropeado: El motor empleado presenta un número de jaulas rotas.

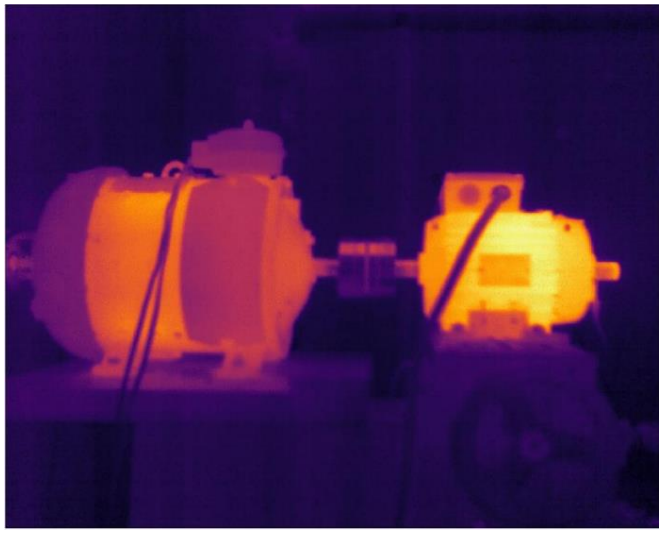

Ilustración 20. Imagen de un motor funcionando correctamente

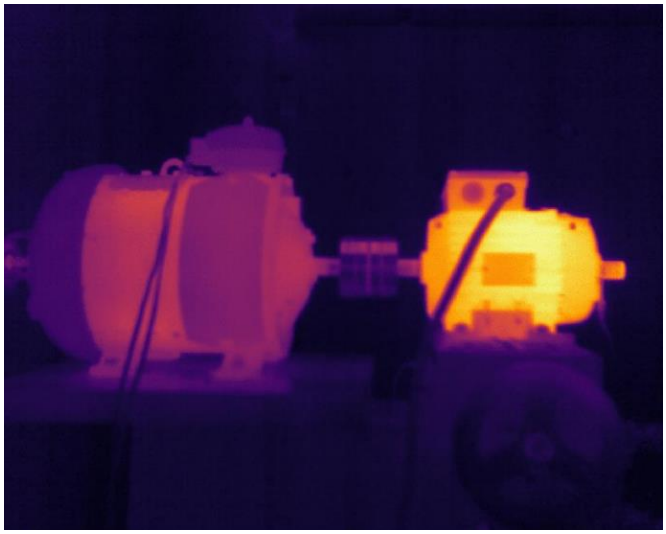

Ilustración 21. Imagen de un motor desalineado

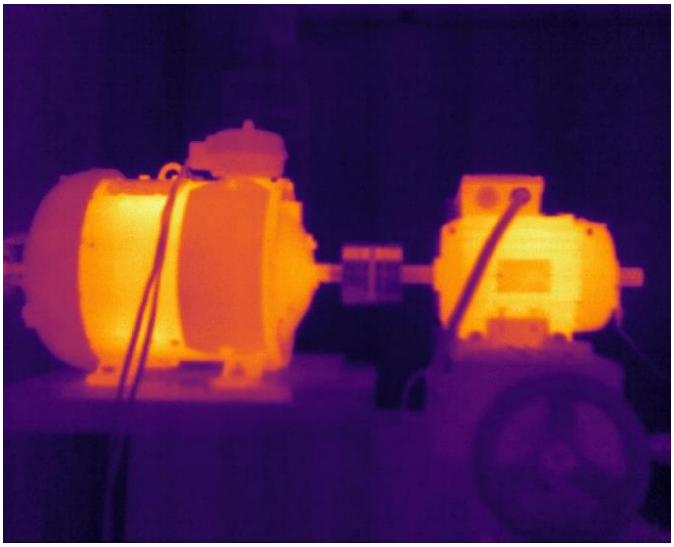

Ilustración 22. Imagen de un motor con alguna jaula rota

A la hora de manejar estos datos y poder ser leídos de la mejor manera por la red neuronal durante su entrenamiento, se decide ordenarlos de la siguiente manera:

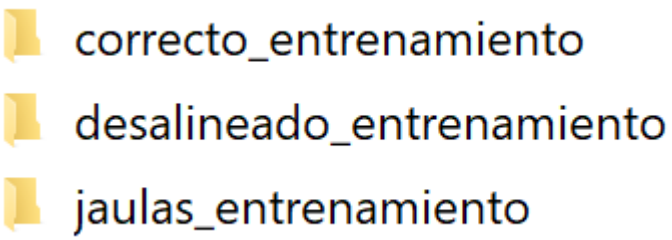

Ilustración 23. Disposición de los datos de entrenamiento

En cada una de esas carpetas se encuentran las imágenes. Para los datos de validación se sigue la misma estructura.

4.4 DISEÑO DE LA RED NEURONAL

En este capítulo se van a explicar las distintas partes del código inicial que se escribe para generar la red neuronal. A partir de este código se irán generando distintas versiones hasta obtener una red que trabaje de la manera más precisa posible.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os
import numpy as np
import matplotlib.pyplot as plt
```

En las primeras líneas de código se importan las librerías y herramientas que se explicaron en el capítulo 4.2. De esta manera se prepara y configura el entorno para proceder a el desarrollo de la red neuronal.

```
datos_entrenamiento = r'C:\Users\usuario\Desktop\TFM\Imagenes\Entrenamiento'
datos_validacion = r'C:\Users\usuario\Desktop\TFM\Imagenes\Validacion'

carpeta_correcto_entrenamiento =
r'C:\Users\usuario\Desktop\TFM\Imagenes\Entrenamiento\correcto_entrenamiento'
carpeta_desalineado_entrenamiento =
r'C:\Users\usuario\Desktop\TFM\Imagenes\Entrenamiento\desalineado_entrenamient
o'
carpeta_jaulas_entrenamiento =
r'C:\Users\usuario\Desktop\TFM\Imagenes\Entrenamiento\jaulas_entrenamiento'

carpeta_correcto_validacion =
r'C:\Users\usuario\Desktop\TFM\Imagenes\Validacion\correcto_validacion'
carpeta_desalineado_validacion =
r'C:\Users\usuario\Desktop\TFM\Imagenes\Validacion\desalineado_validacion'
carpeta_jaulas_validacion =
r'C:\Users\usuario\Desktop\TFM\Imagenes\Validacion\jaulas_validacion'
```

La siguiente parte del código es en la que se definen los directorios donde se encuentran las distintas imágenes del *dataset*:

- *Datos_entrenamiento*: Directorio de la carpeta donde se guardan los datos que se van a emplear para el entrenamiento de la red.
- *Datos_validacion*: Directorio de la carpeta donde se guardan los datos que se van a emplear en la validación del entrenamiento de la red neuronal.
- Las siguientes tres líneas de código son los subdirectorios a las carpetas de las distintas clases para el entrenamiento. Así, hay una carpeta para imágenes

correctas, otro para imágenes de motores desalineadas y la última carpeta contiene las imágenes de los motores con alguna de sus jaulas rotas.

- Las últimas tres líneas son los subdirectorios a las carpetas de las distintas clases para la validación de los datos.

```
num_correcto_entren = len(os.listdir(carpeta_correcto_entrenamiento))
num_desalineado_entren = len(os.listdir(carpeta_desalineado_entrenamiento))
num_jaulas_entren = len(os.listdir(carpeta_jaulas_entrenamiento))
num_correcto_val = len(os.listdir(carpeta_correcto_validacion))
num_desalineado_val = len(os.listdir(carpeta_desalineado_validacion))
num_jaulas_val = len(os.listdir(carpeta_jaulas_validacion))
total_entrenamiento = num_correcto_entren + num_desalineado_entren +
num_jaulas_entren
total_val = num_correcto_val + num_desalineado_val + num_jaulas_val
```

A continuación, se obtiene el número de imágenes en cada directorio y subdirectorio tanto de los datos de entrenamiento como de los datos de validación; así como ese número de datos por clase de imagen, es decir, imágenes correctas para entrenamiento, imágenes correctas para validación, imágenes desalineadas para entrenamiento, etc. Para ello se emplean las funciones 'os. listdir ()' y 'len ()'.

Así, con la primera de las funciones se obtiene una lista de los archivos del directorio que se le pasa como argumento y la segunda se encarga de determinar el número de imágenes que hay en esa lista.

```
TAMANO_LOTE = 100
TAMANO_IMG = 128
```

En estas dos líneas se definen dos variables que son fundamentales a la hora de entrenar el modelo:

1. TAMANO_LOTE: Determina el número de imágenes que se toman en cada época de entrenamiento. Es muy probable que esta variable se modifique alguna vez a la hora de intentar hacer el modelo más preciso. Un mayor número de imágenes hace el proceso de entrenamiento más rápido, pero puede requerir más memoria RAM.

2. TAMANO_IMG: Es el número de pixeles que se establece a la hora de tratar las imágenes. En este caso se van a emplear como entrada imágenes de 128x128 pixeles.

```
image_gen_entrenamiento = ImageDataGenerator(
      rescale=1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
```

```
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

Estas líneas de código son un generador de nuevas imágenes a partir de las imágenes del *dataset*. A esto se le conoce como aumento de datos. Esta técnica consiste en transformar de manera aleatoria las imágenes que se van a emplear en el entrenamiento, generando nuevas imágenes que servirán a su vez para el entrenamiento del modelo. De esta manera se consigue aumentar el número de datos del *dataset* lo que mejorará la capacidad de la red neuronal.

Normalmente se modifican las imágenes con rotaciones, giros tanto en horizontal como en vertical, cizallando la imagen, etc. Además, de esta manera se consigue simular ciertas variaciones que pueden aparecer en los datos que se introducirán en la red en un futuro para predecir el funcionamiento de los motores: cambios en la iluminación, imágenes borrosas o imágenes no perfectamente verticales; por ejemplo.

En el modelo que se está desarrollando, estas son las modificaciones que se van a aplicar a las imágenes:

- *Rescale*: Normaliza el valor de los pixeles escalándolos entre 0 y 1. Esto permitirá a la red trabajar mejor al estar en un rango menor los valores.
- *Rotation_range*: Indica el rango de ángulos en que las imágenes van a ser rotadas. Para este modelo las imágenes se van a rotar entre 40 y -40 grados.
- *Width_shift_range*: Rango en que las imágenes van a ser desplazadas horizontalmente. Este valor debes estar siempre entre 0 y 1.
- *Height_shift_range*: Rango en que las imágenes van a ser desplazadas verticalmente.
- *Shear_range*: Con este parámetro se va a controlar el cizallamiento que se aplica a las imágenes.
- *Zoom_range*: Es el zoom que se aplica a los datos. En este caso se va a comenzar aplicando un zoom en el rango de -20% a +20%.
- *Horizontal_flip*: Se queda indicado que las imágenes pueden ser espejadas con una probabilidad del 50%.

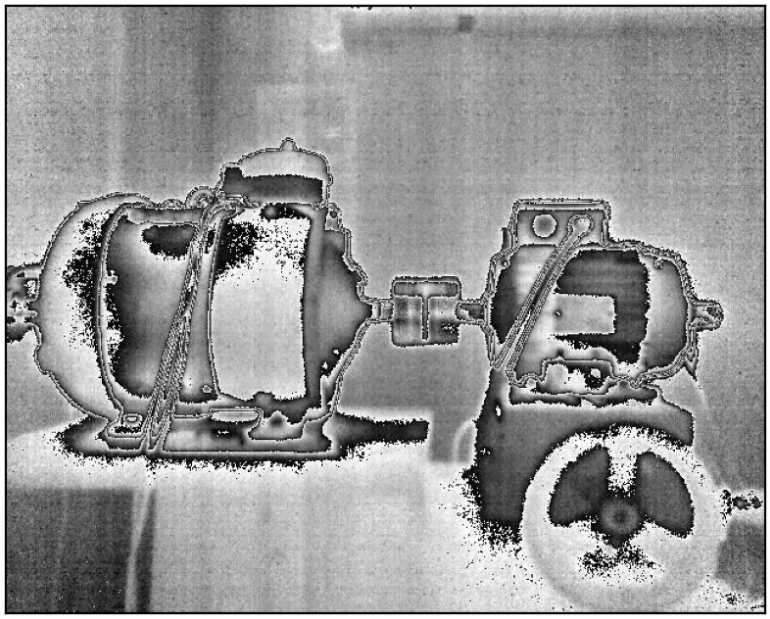

Ilustración 24. Imagen original sin aumento de datos

Ilustración 25. Imagen con aumento de datos

```
data_gen_entrenamiento =
image_gen_entrenamiento.flow_from_directory(batch_size=TAMANO_LOTE,
    directory=datos_entrenamiento,
    shuffle=True,
    target_size=(TAMANO_IMG,TAMANO_IMG),
    class_mode='sparse')
```

Con estas líneas lo que se lleva a cabo es el aumento de datos. Con la función *flow_from_directory* se cargan las imágenes de un directorio (que en este caso es el de

los datos de entrenamiento) en lotes de 100, que fue lo que se fijó con la variable *TAMANO_LOTE*. Con el argumento *shuffle* lo que se hace es mezclar las imágenes para que la red no aprenda un orden específico de las mismas. *Target_size* es el argumento con el que se indica el tamaño de las imágenes, que en este primer caso será de 128x128 pixeles como quedo establecido con la variable *TAMANO_IMG*. Por último, con el argumento *class_mode* se indica el número de etiquetas para el modelo. Para el actual se indica la clase 'sparse' y a cada clase le corresponderá un entero.

```
image_gen_val = ImageDataGenerator(rescale=1./255)

data_gen_validacion =
image_gen_val.flow_from_directory(batch_size=TAMANO_LOTE,
    directory=datos_validacion,
    target_size=(TAMANO_IMG, TAMANO_IMG),
    class_mode='sparse')
```

Con estas líneas se preparan las imágenes de validación para poder llevar a cabo de la mejor manera su evaluación en el modelo. Como se puede observar el código es idéntico al empleado con los datos de entrenamiento a excepción de las líneas de aumento de datos. Esto se debe a que esta técnica generalmente solo se aplica a los datos de entrenamiento.

```
modelo = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
input_shape=(TAMANO_IMG, TAMANO_IMG, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])
```

En este fragmento de código es en donde queda definida la arquitectura de la red neuronal convolucional. Para el desarrollo del modelo se ha empleado el modelo secuencial de *Keras.* Cada línea es una de las capas que se emplea en el modelo y como se explicó en el capítulo 2.5.2, se han aplicado alternamente capas de convolución y capas

de agrupación entre otras. A continuación, se indican los tipos de capas que han sido usadas para configurar la red neuronal:

1. *Conv2D*: Es una capa de convolución. Los filtros o núcleos que se van a aplicar a las imágenes son 32 y cada filtro tendrá un tamaño de 3x3. Además, se utiliza la función de activación ReLU para introducir la no linealidad. Como se trata de la primera capa de la red neuronal, se debe indicar como son las imágenes que van a ser procesadas. Esto se hace con el argumento *input_shape* que dice que las imágenes van a tener un tamaño de TAMANO_IMG x TAMANO_IMG pixeles y que las imágenes van a estar en color (esto es por el número 3; si fuesen en blanco y negro ahí se debería escribir un 1).

2. *MaxPooling2D*: Se trata de una capa de agrupación. El argumento de esta capa se ha configurado para que tome los pixeles en matrices de 2x2 y así reducir a la mitad las imágenes en cada paso por estas capas.

3. A continuación, se incluyen más conjuntos de capas de convolución más capas de agrupación. Las capas de convolución varían en el número de filtros que se aplican a las imágenes, mientras que todas las capas de agrupación son idénticas.

4. *Dropout*: Esta capa es muy importante en el entrenamiento de las redes neuronales. Sirve para "desconectar" un porcentaje de neuronas y al igual que el aumento de datos, es una técnica que se emplea para mejorar el rendimiento del modelo. Al desconectar una parte de las neuronas aleatoriamente, obligas a que el resto aprendan de una manera más robusta ya que tendrán que hacer el doble de trabajo por ejemplo en el caso de que se "apaguen" el 50% de las neuronas en un paso del entrenamiento. En el caso de este primer modelo, se indica con el 0.5 que tiene esta capa como argumento que se van a desconectar la mitad en cada época de entrenamiento.

5. *Flatten*: Es una capa que se emplea para que la información entre a la última capa del modelo.

6. *Dense*: Son las dos últimas capas de la red neuronal. Se necesitan emplear dos capas densas ya que se necesitan reducir las dimensiones de los datos que se aprenden y después poder clasificarlos, el primer argumento de la última capa densa es 3 ya que la red debe clasificar las imágenes en tres grupos. Además, se emplean dos funciones de activación distintas en cada capa de salida.

```
modelo.compile(optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])
```

Una vez definido el modelo, se debe compilar. Para ello se va a utilizar el optimizador Adam siendo el primer argumento de la función. Se ha decidido usar este optimizador como primera opción ya que es el más empleado por su rapidez y eficacia. Como segundo argumento se debe usar una función de pérdida para así poder evaluar como de bien se comporta el modelo durante el entrenamiento. Al quererse resolver un problema de clasificación multiclase, se emplea la función de entropía cruzada. Por último, se debe monitorizar el rendimiento del modelo durante sus distintas etapas y para ello se decide ver la *accuracy*. Con esta función se va a poder ver como de preciso es el modelo en cada una de las etapas al mostrar la fracción de imágenes clasificadas de manera correcta.

```
print("Entrenando modelo...");
epocas=60
history = modelo.fit_generator(
    data_gen_entrenamiento,
    steps_per_epoch=int(np.ceil(total_entrenamiento / float(TAMANO_LOTE))),
    epochs=epocas,
    validation_data=data_gen_validacion,
    validation_steps=int(np.ceil(total_val / float(TAMANO_LOTE)))
)
```

En este bloque se lleva a cabo el entrenamiento del modelo. Cada parte del código hace lo siguiente:

1. En la segunda línea se definen las épocas de entrenamiento. Este parámetro se irá modificando en las distintas pruebas de la red para ir ajustando el número de épocas mínimas necesarias. Se decide iniciar con 60 épocas de entrenamiento.

2. *history = modelo.fit_generator*: Con la variable *history* se guarda el historial de entrenamiento de la red neuronal. Se va a emplear el método *fit_generator* para el entrenamiento y cuyos argumentos se explican a continuación.

3. El primer argumento que se le pasa son los datos que se guardaron en la variable *data_gen_entrenamiento* que eran las imágenes con el aumento de datos.

4. *Steps_per_epoch*: Esta variable se emplea para guardar el número de pasos en cada época de entrenamiento del modelo. En cada paso se debe tomar el tamaño del lote que se guardó en la variable TAMANO_LOTE y como se deben usar todas las imágenes en cada época de entrenamiento, el número de pasos en cada época será el número total de datos de entrenamiento entre el tamaño del lote. Para asegurar que se usen todas las imágenes en cada paso se redondea al número mayor con *np.ceil*.

5. *Epochs*: El número de épocas de entrenamiento que quedaron definidas anteriormente.

6. *Validation_data*: Variable en la que se almacenan las imágenes que se van a usar en la validación de la red neuronal.

7. *Validation_steps*: Es el mismo procedimiento que con los pasos de entrenamiento, pero en este caso en la validación.

```
modelo.save('modelo_motores.h5')
print("Modelo guardado!");

# Mostrar la accuracy y la pérdida
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.ylim([0, 2])
plt.legend(loc='upper right')
plt.show()
```

Este es el último fragmento del código empleado para el diseño y entrenamiento de la red neuronal. En el primer bloque se guarda el modelo entrenado para así poder usarlo más adelante a la hora de realizar las predicciones con imágenes que no han sido empleadas durante el entrenamiento.

Los dos siguientes bloques muestran la *accuracy* del entrenamiento y la función de pérdida. Para ello se utiliza Matplotlib que imprime por pantalla las dos gráficas que se quieren visualizar. Con la *accuracy* se conseguirá ver qué porcentaje de las muestras han sido etiquetadas de manera correcta y con la función de pérdida se mide como de erróneas son las predicciones realizadas.

4.5 ENTRENAMIENTO DE LA RED NEURONAL

4.5.1 MODELO 1

En este primer entrenamiento de le red, los parámetros empleados y que son susceptibles de ser cambiados para ir mejorando los resultados del entrenamiento son los siguientes:

- TAMANO_LOTE: 100
- TAMANO_IMG: 128
- Número de capas de convolución y agrupación: 4
- Número de neuronas en cada capa de convolución: 32, 64, 128,128
- Épocas de entrenamiento: 60

Con estos parámetros se entrena la red, obteniéndose 48 pasos por época de entrenamiento y con una media de 3 segundos por paso. Tras la finalización del entrenamiento, el resultado no es óptimo ya que la máxima *accuracy* alcanzada por el modelo es de un 38%, así como la función de pérdida está constantemente por encima de 1 desde el inicio de esta prueba. Además, se aprecia que el modelo no mejora su rendimiento desde la época 4 en adelante. Algunas de las conclusiones que se pueden sacar de esta primera prueba son: el número de épocas de entrenamiento es excesivo ya que el modelo no necesita tantas para mejorar su *accuracy* y que es necesario cambiar algunos de los parámetros para mejorar los resultados de la red neuronal.

A continuación, se muestran las gráficas de *accuracy* y pérdida de la red neuronal durante este primer entrenamiento:

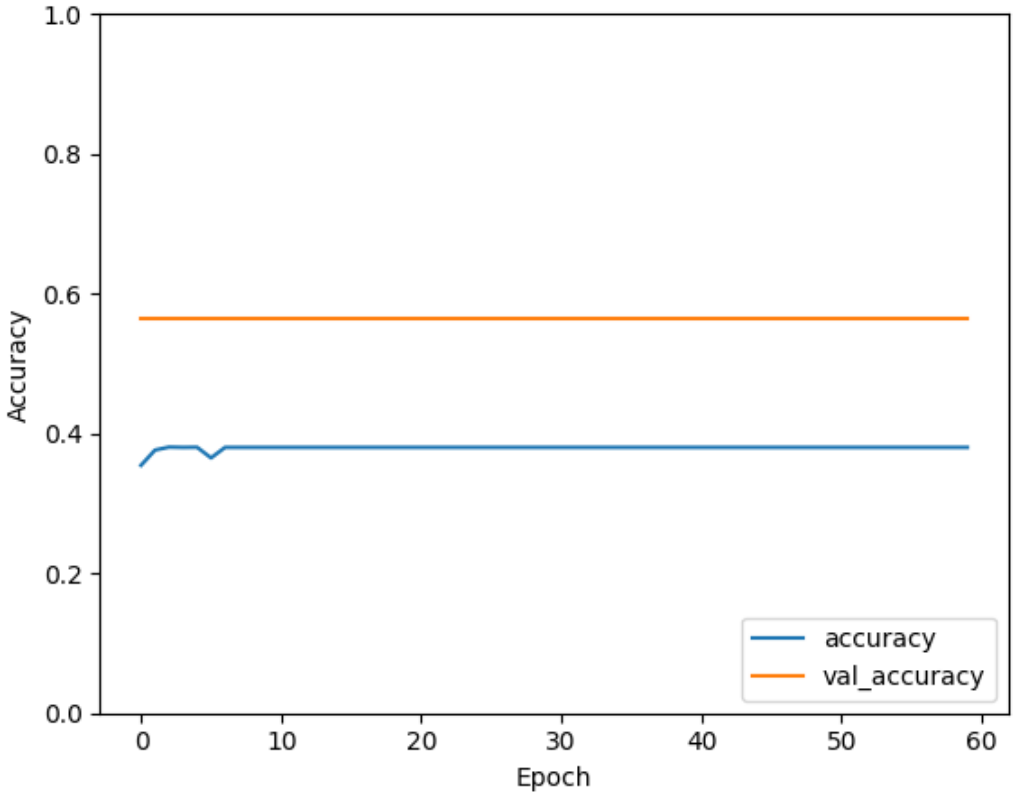

Ilustración 26. Accuracy del modelo 1

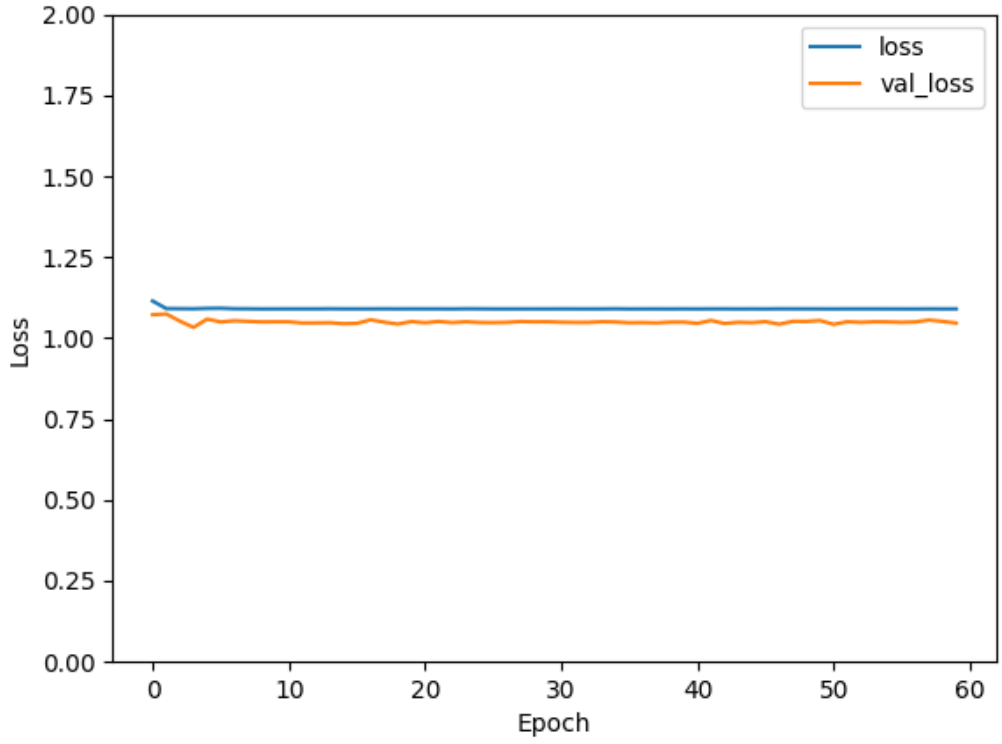

Ilustración 27. Función de pérdida del modelo 1

4.5.2 MODELO 2

Para tratar de optimizar el modelo, se decide leer de una manera relativa las rutas de los archivos de entrenamiento y validación:

```
base_dir = r'C:\Users\usuario\Desktop\TFM\Imagenes'
train_dir = os.path.join(base_dir, 'Entrenamiento')
validation_dir = os.path.join(base_dir, 'Validacion')
```

Además, se cambia la variable TAMANO_LOTE a 32 así como la manera en que se imprimen por pantalla las gráficas, de manera que los valores de los ejes se ajusten a los valores obtenidos.

Los resultados no vuelven a ser los esperados, ya que no ha mejorado ni la *accuracy* ni la pérdida y el número de épocas vuelve a estar sobredimensionado al no variar estos parámetros prácticamente desde la época 4 del entrenamiento. A su vez, los tiempos de compilación son muy elevados. Las gráficas de este modelo quedan de la siguiente manera:

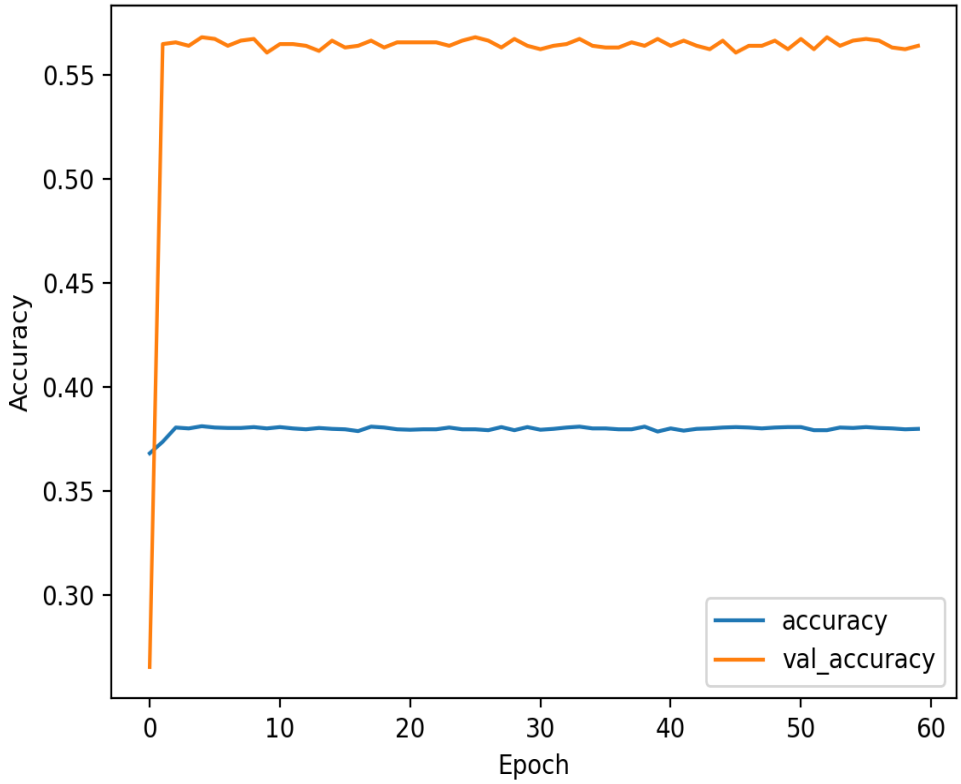

Ilustración 28. Accuracy del modelo 2

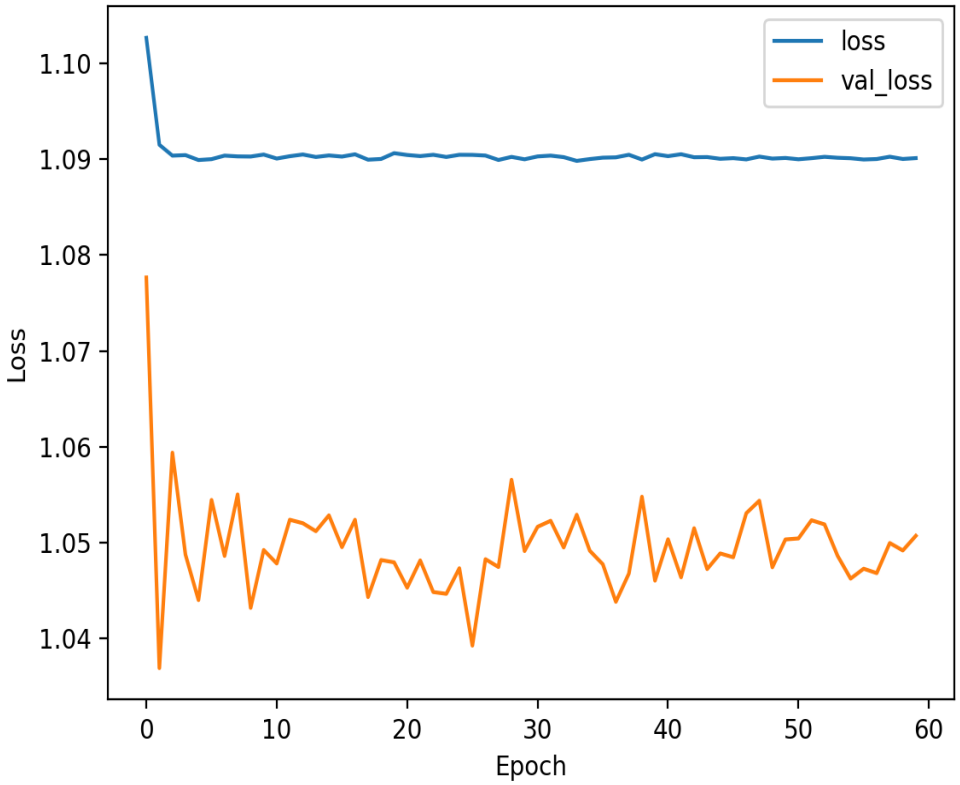

Ilustración 29. Función de pérdida del modelo 2

4.5.3 MODELO 3

En el siguiente modelo se introducen 4 cambios y se mantiene la manera relativa de leer las carpetas:

- TAMANO_IMG: 250
- TAMANO_LOTE: 100
- Épocas de entrenamiento: 10
- Se elimina del modelo el aumento de datos, ya que a la hora de hacer ese aumento con las imágenes a color se observa que no se realiza de manera correcta. Además, como el número de datos de entrenamiento es aproximadamente 4000 para estas primeras pruebas en las que todas las imágenes empleadas van a ser del *dataset*, se decide que no es necesario aplicarlo de momento.

Los resultados obtenidos son idénticos a los dos modelos anteriores. Por lo tanto, se concluye que muy probablemente la red neuronal no está leyendo de manera correcta las imágenes al ser muy extraño alcanzar *accuracys* idénticas con modelos distintos. También se observa que no es óptimo realizar estas primeras pruebas, sin datos con una precisión muy elevada, con las imágenes en resolución 250x250 pixeles ya que el tiempo de entrenamiento del modelo es muy superior a las imágenes de 100x100 pixeles.

4.5.4 MODELO 4

Como las pruebas que se hicieron de las imágenes en blanco y negro aplicándoles el aumento de datos (*Ilustración 24* e *Ilustración 25*) fueron satisfactorias, se decide probar a introducir las imágenes en este formato. También se decide volver al formato de imágenes en 100x100 pixeles por el tiempo de procesamiento del modelo y se mantiene la red neuronal sin aumento de datos por el momento para tratar de simplificar al máximo la complejidad del código hasta obtener resultados más satisfactorios. Por lo tanto, las imágenes ahora se introducen de esta manera:

```
data_gen_entrenamiento = image_gen_entrenamiento.flow_from_directory(
    batch_size=TAMANO_LOTE,
    directory=datos_entrenamiento,
    shuffle=True,
    color_mode='grayscale',
    target_size=(TAMANO_IMG, TAMANO_IMG),
    class_mode='sparse')
```

Los resultados que se obtienen tras el entrenamiento de la red son los esperados:

- La accuracy alcanzada tras la séptima época de entrenamiento es de aproximadamente un 100%
- La función de pérdida es cercana a cero tras la séptima época de entrenamiento también.

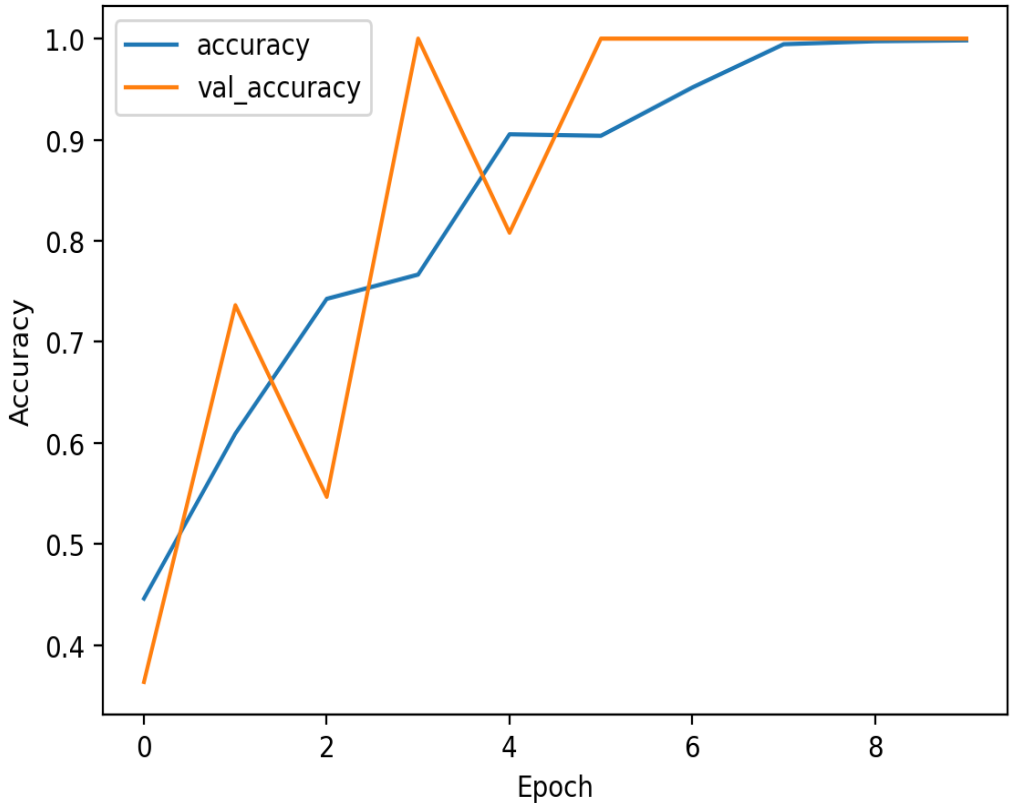

Ilustración 30. Accuracy del modelo 4

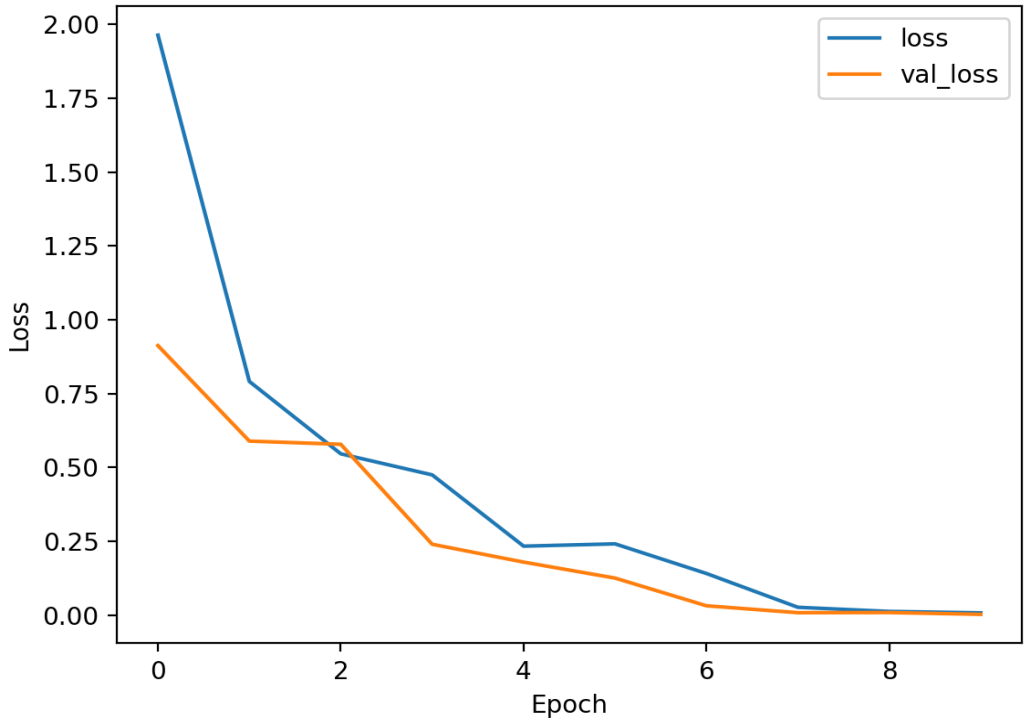

Ilustración 31. Función de pérdida del modelo 4

Por lo tanto, las imágenes se procesan mejor si se introducen en blanco y negro. Esto se puede deber a diversas causas como son:

- Al eliminar los canales de color, la complejidad de las imágenes se reduce y la red neuronal se puede centrar en las características importantes para la clasificación de los datos.
- A su vez, también tiene que procesar menos información que con las imágenes a color.
- La iluminación de la sala no va a afectar a las imágenes en blanco y negro, pero sí influía en las imágenes a color.

4.6 PRUEBAS DE LA RED NEURONAL

4.6.1 PRUEBAS CON DATOS DEL DATASET

Del *dataset* que se ha empleado para realizar el entrenamiento de la red neuronal, se reservaron una serie de imágenes de cada categoría para poder hacer las pruebas del modelo y así poder ver si funciona. Tras introducir una serie de imágenes en el modelo, se verifica que distingue perfectamente este tipo de datos con una elevada precisión. A continuación, se muestra la terminal para un caso:

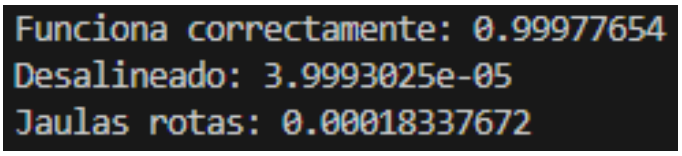

Ilustración 32. Motor funcionando correctamente

Se puede apreciar que, aunque no afirma al 100% que el motor esté funcionando correctamente, da un porcentaje muy elevado y se considera que es más que suficiente para poder identificar como está funcionando el mismo. Este resultado se obtiene de manera sistemática con cualquier categoría y cualquier imagen del banco de datos que se ha estado empleando hasta el momento.

4.6.2 PRUEBAS CON DATOS QUE NO SON DEL DATASET

Para saber si el modelo funciona de manera correcta en cualquier circunstancia, es necesario probar si identifica estos problemas con imágenes que están tomadas en otras condiciones. Por ello, se toma otro *dataset* en el que las imágenes se dividen en motores funcionando correctamente, motores que tienen fallos en el estator y motores que tienen fallos en el rotor (no se hace la diferencia entre motores con jaulas rotas y motores

desalineados). Obviamente, como los fallos en el estator no han sido considerados a la hora de entrenar el modelo, es normal que no los identifique; pero las imágenes de motores funcionando correctamente sí deberían ser clasificadas de manera correcta. Los fallos en el rotor no tienen por qué ser identificados ya que se trata de casos en los que se ha quedado atascado y este fallo no se ha tenido en cuenta en el entrenamiento del modelo.

Tras analizar algunas imágenes, se puede observar que la red no identifica de manera 100% fiable si los motores funcionan correctamente. Aunque acierta en todas sus predicciones, no lo hace con una precisión elevada.

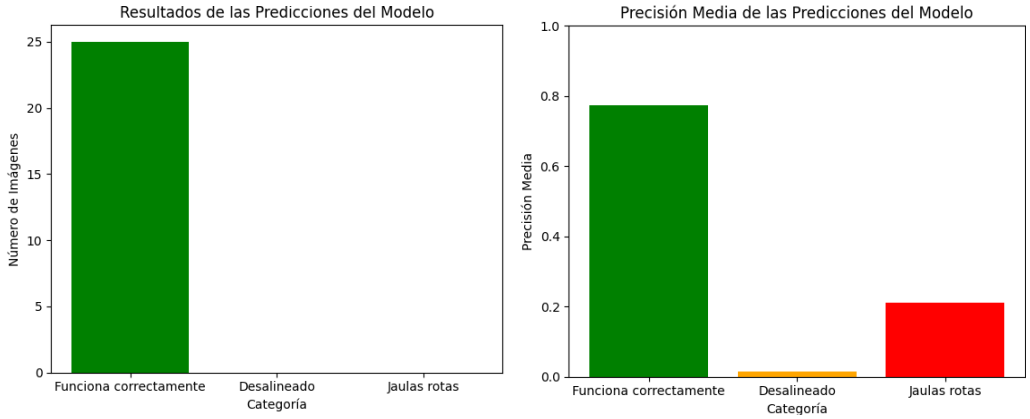

Ilustración 33. Resultados y precisión media de la predicción de motores funcionando correctamente

Esto se puede deber a que se eliminó del modelo el aumento de datos, que era la manera en que se generaban nuevas imágenes para que la red neuronal no se adecuase a unas determinadas condiciones.

A la hora de probar con las imágenes de fallos en el rotor, los resultados son muy malos ya que el modelo no es capaz de identificar este tipo de imágenes ni como motor desalineado ni como motor con jaulas rotas. Es cierto que la red no está preparada para identificar fallos en el rotor que no sean los dos mencionados anteriormente, y este nuevo *dataset* introduce casos en que el rotor está atascado. No obstante, se va a intentar introducir de nuevo el aumento de datos en el modelo para así tratar que se identifiquen el mayor número de casos posibles.

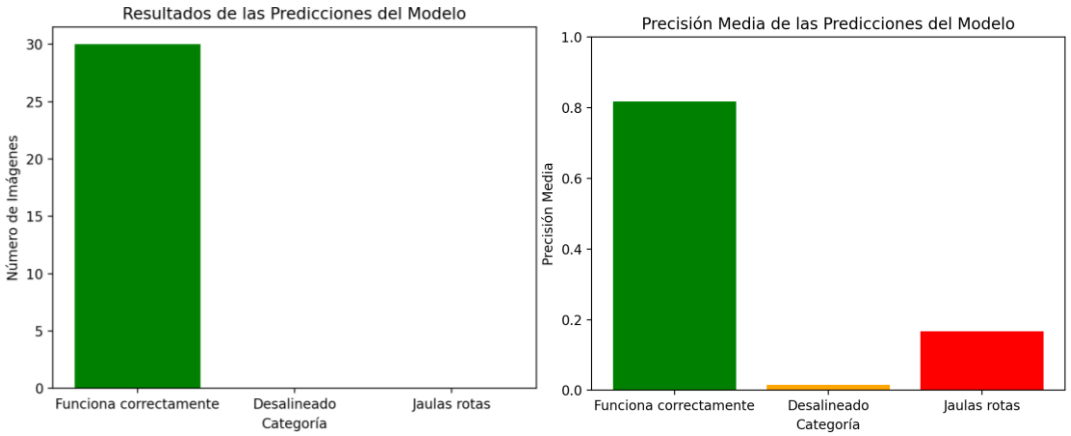

Ilustración 34. Resultados y precisión media de la predicción de motores con fallos en el rotor

4.6.3 MODELO 5

En este modelo, como se ha comentado, se va a volver a incluir el aumento de datos. Para ello se comienza introduciendo el bloque de aumento de datos del apartado 4.4. Además, también se le incluyen variaciones en el brillo a las imágenes para tratar que sean los más heterogéneas posibles.

Tras realizar un par de pruebas de este nuevo modelo, surge un problema con el sobreajuste de la red neuronal. Esto consiste en que la red aprende de manera demasiado precisa los datos, lo que la lleva a identificar también el ruido de las imágenes y por tanto no es capaz de clasificar bien los datos de validación. Es por ello por lo que se incluyen algunas técnicas para tratar de minimizar ese sobreajuste.

Primero se aplica la regularización L2 a los pesos del modelo. Esto se aplica a la función de pérdida del modelo en las capas de convolución, penalizando los pesos más grandes. Cuanto mayor sea el valor, mayor será la penalización que se aplica. En la red neuronal aparece en las capas de convolución de la siguiente manera:

```
tf.keras.layers.Conv2D(64,(3,3),activation='relu',kernel_regularizer=tf.keras.
regularizers.l2(0.001))
```

Se define una tasa de aprendizaje del optimizador Adam. Con este parámetro se consigue que los ajustes que realiza este sean más precisos al tener un menor valor:

```
modelo.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001)
```

Además, se incluye el *early stopping* que es una técnica para parar el entrenamiento del modelo cuando este ya no está aprendiendo más. Como el sobreajuste se ve en los

datos de validación, se define en el código de la red neuronal que el entrenamiento se pare cuando la pérdida de los datos de validación no mejore durante 10 épocas.

```
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=10, restore_best_weights=True)
```

Tras introducir estas técnicas en el código, se consigue subsanar parcialmente el sobreajuste, aunque no totalmente. Además, se han modificado los dos siguientes parámetros para tratar de hacer más preciso el modelo, aunque aumenta el tiempo que tarda la red neuronal en ser entrenada:

- TAMANO_LOTE: 32
- TAMANO_IMG: 200

Se consigue alcanzar una elevada *accuracy* tras 70 etapas de entrenamiento (el *early stopping* para el entrenamiento en esta época) con una *accuracy* inferior en los datos de validación durante las últimas épocas. La pérdida del modelo tarda mucho en mejorar, pero la alcanzada finalmente es aceptable. A continuación, se muestras las gráficas de estos dos parámetros durante el entrenamiento. Se puede apreciar que son más inestables que para los modelos sencillos que se programaron en el apartado anterior:

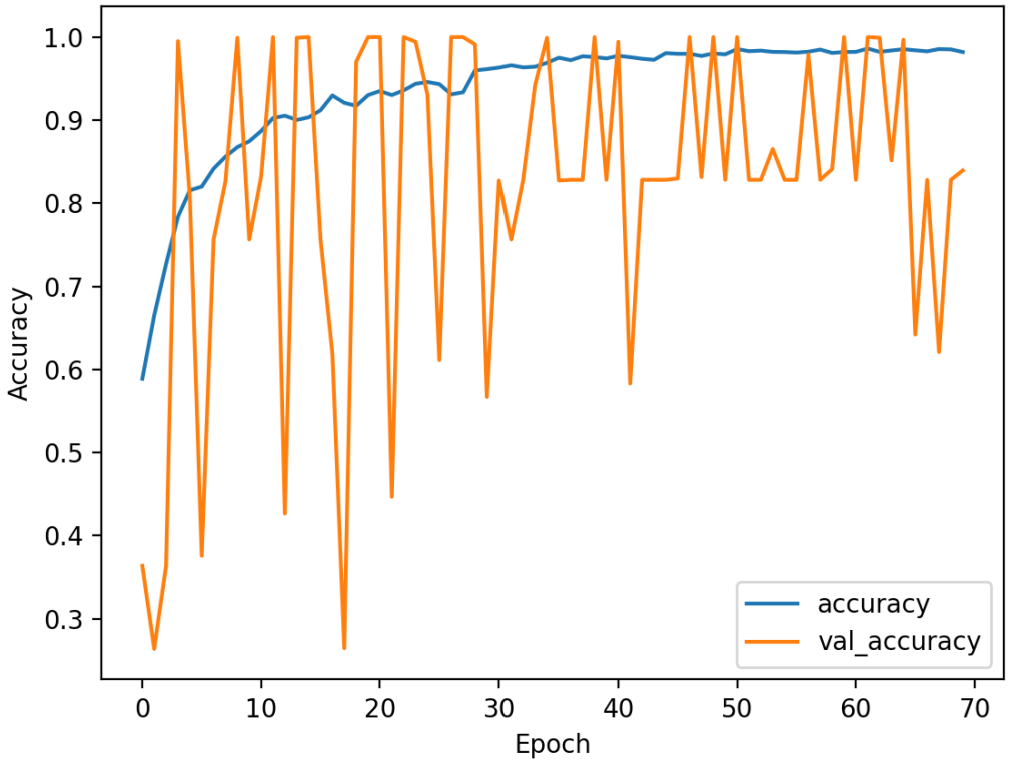

Ilustración 35. Accuracy del modelo 5

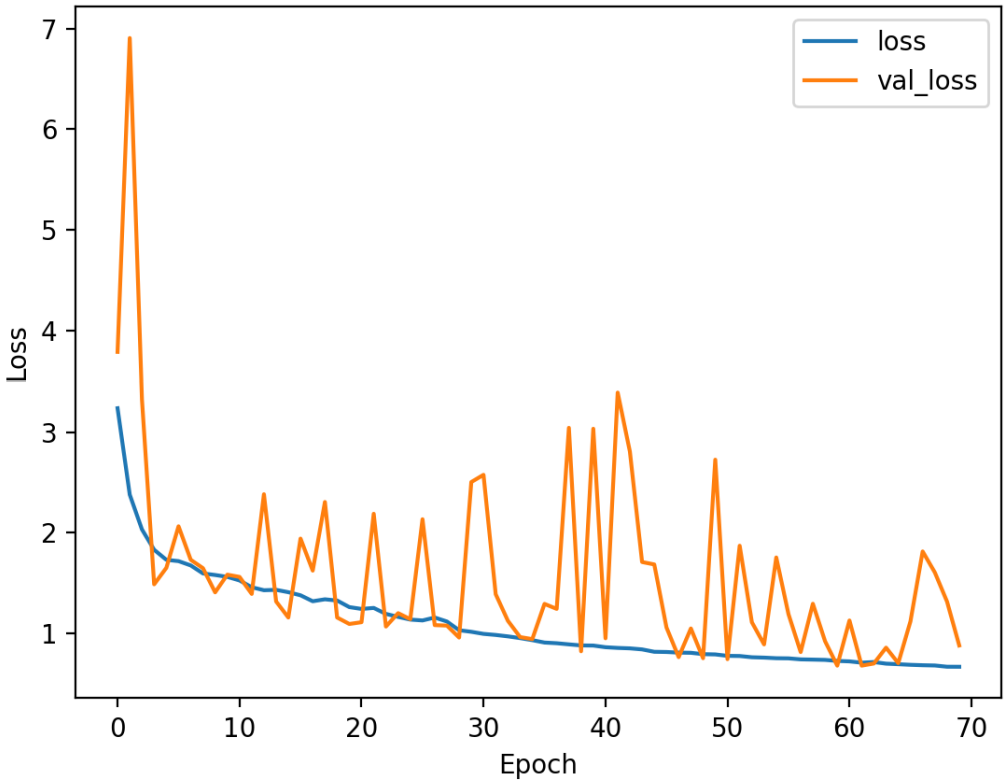

Ilustración 36. Función de pérdida del modelo 5

4.7 RESULTADOS

Una vez que se han programado y entrenado las dos redes neuronales, se han empleado distintas métricas para evaluar como de buena es cada una y cómo se comportan con distintos datos. Así, se han evaluado los modelos con las imágenes de test y con las imágenes de test aplicándoles el aumento de datos obteniéndose los siguientes resultados:

Resultados con datos de entrenamiento				
Modelo	Accuracy	Precision	Recall	F1 Score
Datos de Test sin Aumento de Datos				
Modelo sin A.D.	1,00	1,00	1,00	1,00
Modelo con A.D.	1,00	1,00	1,00	1,00
Datos de Test con Aumento de Datos				
Modelo sin A.D.	0,202	0,757	0,202	0,183
Modelo con A.D.	0,917	0,938	0,917	0,923

Tabla 3. Resultados obtenidos con el set de test

- *Accuracy*: Es la métrica que se ha usado durante todo el proyecto y representa el porcentaje de imágenes que han sido clasificadas de manera correcta frente al total.

- *Precision*: Nos muestra cuanto se aproxima una predicción al valor real, es decir, mide como de precisa es la red a la hora de predecir casos positivos. Es el cociente de los casos positivos bien clasificados entre las predicciones positivas (predicciones positivas mas falsos positivos).

- *Recall*: Esta métrica indica la proporción de verdaderos positivos sobre el total de casos que deberían haber sido clasificados como positivos. Se obtiene con el cociente de predicciones positivas entre la suma de predicciones positivas mas los falsos negativos.

- *F1 Score*: Es la media armónica entre la *precisión* y el *recall*.

Se observa que tanto el modelo con aumento de datos como el modelo sin aumento de datos, clasifican el 100% de las imágenes bien cuando se tratan sin aumento de datos. Sin embargo, cuando esas mismas imágenes se modifican se puede apreciar que la red neuronal que ha sido entrenada sin aumento de datos no es capaz de clasificar de manera correcta las imágenes. Es en estos resultados donde se ve lo necesaria que es aplicar esta técnica, ya que el modelo es capaz de predecir imágenes que han sido tomadas en las mismas condiciones que el set de entrenamiento y validación, pero se vuelve totalmente inútil cuando se introducen pequeñas variaciones de dichos datos.

Por otro lado, los resultados de la red neuronal con aumento de datos, sin ser extremadamente buenos, se acercan más a un modelo preciso y que podría empezar a emplearse en la predicción de los motores eléctricos.

Al obtener la red con A.D. una *accuracy* del 92% y siendo muy superior al 20% de la otra red, se puede apreciar que la red es mucho más robusta y que es capaz de generalizar mejor los datos que difieran de los originales. Si analizamos la *precision*, aunque es la métrica más parecida para los dos casos, el segundo modelo es mejor a la hora de evitar falsos positivos

A continuación, se muestran las matrices de confusión de los dos modelos cuando se han hecho las pruebas con los datos modificados. Estas matrices representan el número de imágenes que han sido clasificadas correctamente:

		Predicciones		
		Correcto	**Desalineado**	**Jaulas Rotas**
Real	**Correcto**	68	0	38
	Desalineado	345	47	123
	Jaulas Rotas	69	0	31

Tabla 4. Matriz de confusión del modelo sin A.D.

		Predicciones		
		Correcto	**Desalineado**	**Jaulas Rotas**
Real	**Correcto**	99	0	7
	Desalineado	40	475	0
	Jaulas Rotas	13	0	87

Tabla 5. Matriz de confusión del modelo con A.D.

A la vista de los resultados obtenidos con estas matrices, se observa que el modelo sin aumento de datos no es capaz de clasificar correctamente las imágenes. Por ejemplo, de las 106 imágenes que existen de un motor funcionando correctamente, tan solo es capaz de identificar 68. Estos errores se vuelven más numerosos para las otras dos clases en que debe clasificar la red neuronal.

Por otro lado, el modelo que ha sido entrenado con amento de datos es capaz, como se esperaba, de clasificar los datos de manera mucho más correcta. Aunque no acierta en todas las predicciones, es capaz de identificar casi todos los casos. Por ejemplo, del caso de motor funcionando correctamente identifica 99 de las 106 imágenes, un número bastante superior al obtenido por la otra red.

Capítulo 5. CONCLUSIÓN

En este trabajo se ha desarrollado la base para un sistema de control de temperatura en la industria 4.0. Este objetivo se nace de la necesidad de realizar un mantenimiento predictivo a una parte fundamental de los sistemas de producción como son los motores eléctricos, con el consiguiente ahorro energético que supondría la suma de mantenerlos en condiciones óptimas de trabajo empleando sistemas de baja potencia.

Inicialmente se pensaba desarrollar por completo esta herramienta, pero debido a la complejidad del problema se optó por implementar una parte de el mismo dejando para futuros proyectos las demás partes que comprendían el sistema. Se configura la red neuronal ya que es la base para poder llevar a cabo el análisis de los motores. Para ello se emplea una base de datos con imágenes térmicas de motores que tienen algún tipo de defecto y que nos va a ayudar a identificar el funcionamiento deficiente antes de que se haya dañado demasiado el mecanismo.

Algunas de las conclusiones que se han obtenido tras desarrollar las dos redes neuronales son las siguientes:

1. Las redes neuronales convolucionales son una fantástica opción para el tratamiento y clasificación de imágenes.

2. Tratar las imágenes en blanco y negro facilita mucho el procesamiento de los datos por lo que se alcanzan resultados más satisfactorios. Esto nos indica que cuanto más facilitemos las cosas a la red neuronal, más satisfactorio y sencillo será su entrenamiento.

3. Tras haber programado dos modelos, el aumento de datos se nos presenta como una técnica que, aunque puede complicar el entrenamiento, mejora el rendimiento de la red cuando se maneja con datos que no han sido tomados en las mismas condiciones que los datos de entrenamiento y validación. Es obvio que en la inmensa mayoría de los casos las condiciones van a ser distintas, por lo que se hace totalmente necesario emplear esta técnica.

4. Queda demostrado que la termografía es un método completamente válido para realizar un mantenimiento predictivo. Esto constituye una gran ventaja frente al resto de métodos que se emplean, ya que es no invasivo.

5. Aunque el modelo funciona muy bien con los dos tipos de fallos que han sido contemplados, no se podría usar todavía en un entorno industrial al poder encontrarnos algún tipo de error no contemplado en este estudio.

Capítulo 6. LÍNEAS FUTURAS

Como trabajos futuros, se debería seguir construyendo el sistema y mejorando la red neuronal que ha sido programada en este. Por ello, se proponen las siguientes líneas futuras:

1. Optimización del modelo programado. Aunque las métricas obtenidas son muy buenas, sería necesario tratar de obtener resultados más cercanos a 1. Se debería hacer pruebas con datos distintos en la red neuronal: modificar las capas de convolución y agrupación, el tamaño de las imágenes, parámetros del aumento de datos… También sería recomendable investigar técnicas que mejoren las métricas como lo fue el aumento de datos en el presente trabajo.

2. Aumento de los datos de entrenamiento y validación. Como en este trabajo solo se han tenido en cuenta dos tipos de fallos en los motores de inducción, se deberían ampliar con otros errores como los incluidos en el segundo *dataset* presente en el *Capítulo 4*. De esta manera se podrían identificar muchos más problemas y el mantenimiento predictivo sería más preciso.

3. Tratar de programar un modelo empleando redes distintas a las CNN. Por ejemplo, se podría construir una red generativa adversaria y ver como se comporta a la hora de resolver este problema.

4. Cargar el modelo en una cámara térmica. Por motivos de tiempo, no se ha podido realizar la compra y posterior implementación del modelo en la cámara térmica M5Stick t-lite. Se podría continuar el trabajo configurando este instrumento para que trabajase primero con redes wifi y después con otros sistemas de baja potencia para así completar el sistema que se pretende armar con el presente proyecto.

5. Uso en ámbitos industriales. Una vez perfeccionado el modelo y habiéndolo cargado en algún dispositivo, se podrían realizar pruebas del mismo en un entorno industrial para ver cómo se comporta y seguir perfeccionándolo.

Capítulo 7. ANEXOS

ANEXO I: PRESUPUESTO

COSTE DIRECTO

COSTE DIRECTO DEL PROYECTO			
CONCEPTO	**TIEMPO EMPLEADO**	**VALOR UNITARIO**	**SUBTOTAL**
RECURSOS HUMANOS	**Horas**	**€/h**	**€**
Investigador Junior	480	20	9.600,00 €
Investigador Senior	68	40	2.720,00 €
LICENCIAS	**Meses**	**€/mes**	**€**
Ms Office	12	7	84,00 €
MATERIAL AMORTIZABLE	**Meses**	**€/mes**	**€**
Ordenador Personal de trabajo	12	33,33	399,96 €
TOTAL SIN IVA			12.803,96 €
IVA (21%)			2.688,83 €
TOTAL			15.492,79 €

Tabla 6. Coste directo del proyecto

COSTE TOTAL

Hay que tener en cuanta los costes indirectos del proyecto, como pueden ser el agua, la electricidad, material necesario… Estos se consideran que son un 25% de los costes directos. Por tanto, el coste total será la suma de los costes directos mas los costes indirectos:

PRESUPUESTO DEL PROYECTO	
COSTE DIRECTO	15.492,79 €
COSTE INDIRECTO	3.873,20 €
COSTE TOTAL	19.365,99 €

Tabla 7. Coste total del proyecto

ANEXO II: ALINEACIÓN CON LOS ODS

El presente trabajo, contribuye al cumplimiento de los siguientes ODS(*Objetivos de Desarrollo Sostenible | Programa De Las Naciones Unidas Para El Desarrollo*, n.d.):

ODS 7: Energía Asequible y No Contaminante

El objetivo principal de este ODS es asegurar que todo el mundo tenga acceso a servicios de energía asequibles, confiables y limpia. Esto conduce a aumentar considerablemente la proporción de energía que proviene de fuentes renovables y duplicar la tasa global de mejora en eficiencia.

Como en este proyecto se van a emplear sistemas de baja potencia en la industria, se promueve la eficiencia energética.

ODS 9: Industria, Innovación e Infraestructuras

Este ODS busca desarrollar infraestructuras que sean fiables, sostenibles y de calidad. Además, se pretende que este desarrollo sea regional y transfronterizo. Otro de sus objetivos es ayudar a pequeñas empresas a tener acceso a servicios financieros, especialmente en aquellos países en desarrollo.

Este ODS es el principal con el que cumple el presente trabajo, al estar centrado en la industria y en la innovación a través del sistema que se desarrolla. Esta propuesta impulsa la adopción de tecnologías avanzadas y aporta una solución innovadora para la industria, que además es incorporada a la Industria 4.0.

ODS 12: Producción y Consumo Responsables

Este objetivo busca reducir la generación de desechos y promover la producción más responsable, entre otras metas.

Al diseñar este sistema e implantarlo en la industria, se consigue disminuir los recursos que se emplean al usar sistemas de baja potencia y aumentar la eficiencia de los procesos; por lo que da pie a una producción responsable.

ODS 13: Acción por el Clima

El ODS 13, busca mitigar los efectos del cambio climático a través de la educación, la incorporación de medidas en planes estratégicos, mejorando la capacidad de adaptación a los nuevos riesgos y promoviendo mecanismos de gestión eficaz en aquellos países

menos adelantados. En resumen, tomar medidas urgentes para promover la sensibilización sobre el cambio climático y su mitigación.

En este Trabajo de Fin de Máster se promueve la eficiencia energética al emplear un sistema de baja energía y por lo tanto se cumple con la parte de mitigación que propone este ODS, al contribuir a reducir las emisiones.

ANEXO III: CÓDIGOS EMPLEADOS

Modelo sin aumento de datos.

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator # type:
ignore
import os
import numpy as np
import matplotlib.pyplot as plt

# Variables de directorio
datos_entrenamiento = r'C:\Users\usuario\Desktop\TFM\Imagenes\Entrenamiento'
datos_validacion = r'C:\Users\usuario\Desktop\TFM\Imagenes\Validacion'

# Número de datos de entrenamiento
num_correcto_entren = len(os.listdir(os.path.join(datos_entrenamiento,
'correcto_entrenamiento')))
num_desalineado_entren = len(os.listdir(os.path.join(datos_entrenamiento,
'desalineado_entrenamiento')))
num_jaulas_entren = len(os.listdir(os.path.join(datos_entrenamiento,
'jaulas_entrenamiento')))
num_correcto_val = len(os.listdir(os.path.join(datos_validacion,
'correcto_validacion')))
num_desalineado_val = len(os.listdir(os.path.join(datos_validacion,
'desalineado_validacion')))
num_jaulas_val = len(os.listdir(os.path.join(datos_validacion,
'jaulas_validacion')))
total_entrenamiento = num_correcto_entren + num_desalineado_entren +
num_jaulas_entren
total_val = num_correcto_val + num_desalineado_val + num_jaulas_val

TAMANO_LOTE = 100
TAMANO_IMG = 100

# Generación de datos de entrenamiento sin aumento
image_gen_entrenamiento = ImageDataGenerator(rescale=1./255)

data_gen_entrenamiento = image_gen_entrenamiento.flow_from_directory(
    batch_size=TAMANO_LOTE,
    directory=datos_entrenamiento,
    shuffle=True,
    color_mode='grayscale',# Las imágenes son en blanco y negro
    target_size=(TAMANO_IMG, TAMANO_IMG),
    class_mode='sparse')

# Generación de datos de validación
image_gen_val = ImageDataGenerator(rescale=1./255)

data_gen_validacion = image_gen_val.flow_from_directory(
    batch_size=TAMANO_LOTE,
    directory=datos_validacion,
    color_mode='grayscale', # Las imágenes son en blanco y negro
    target_size=(TAMANO_IMG, TAMANO_IMG),
```

```
    class_mode='sparse')

# Modelo
modelo = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
input_shape=(TAMANO_IMG, TAMANO_IMG, 1)),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Compilación
modelo.compile(optimizer='adam',
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])

# Entrenamiento del modelo
print("Entrenando modelo...")
epocas = 10
history = modelo.fit_generator(
    data_gen_entrenamiento,
    steps_per_epoch=int(np.ceil(total_entrenamiento / float(TAMANO_LOTE))),
    epochs=epocas,
    validation_data=data_gen_validacion,
    validation_steps=int(np.ceil(total_val / float(TAMANO_LOTE)))
)

print("Modelo entrenado!")

# Guardar el modelo
modelo.save('modelo_motores_bn.h5')
print("Modelo guardado!")

# Visualización de accuracy y pérdida
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Modelo con aumento de datos

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator #type:
ignore
import os
import numpy as np
import matplotlib.pyplot as plt

datos_entrenamiento = r'C:\Users\usuario\Desktop\TFM\Imagenes\Entrenamiento'
datos_validacion = r'C:\Users\usuario\Desktop\TFM\Imagenes\Validacion'

# Número de datos de entrenamiento y validación
num_correcto_entren = len(os.listdir(os.path.join(datos_entrenamiento,
'correcto_entrenamiento')))
num_desalineado_entren = len(os.listdir(os.path.join(datos_entrenamiento,
'desalineado_entrenamiento')))
num_jaulas_entren = len(os.listdir(os.path.join(datos_entrenamiento,
'jaulas_entrenamiento')))
num_correcto_val = len(os.listdir(os.path.join(datos_validacion,
'correcto_validacion')))
num_desalineado_val = len(os.listdir(os.path.join(datos_validacion,
'desalineado_validacion')))
num_jaulas_val = len(os.listdir(os.path.join(datos_validacion,
'jaulas_validacion')))
total_entrenamiento = num_correcto_entren + num_desalineado_entren +
num_jaulas_entren
total_val = num_correcto_val + num_desalineado_val + num_jaulas_val

TAMANO_LOTE = 32
TAMANO_IMG = 200

# Aumento de datos
print("Realizando aumento de datos")
image_gen_entrenamiento = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2],
    fill_mode='nearest'
)

data_gen_entrenamiento = image_gen_entrenamiento.flow_from_directory(
    batch_size=TAMANO_LOTE,
    directory=datos_entrenamiento,
    shuffle=True,
    color_mode='grayscale',
    target_size=(TAMANO_IMG, TAMANO_IMG),
    class_mode='sparse'
)
```

```
# Generación de datos de validación
image_gen_val = ImageDataGenerator(rescale=1./255)

data_gen_validacion = image_gen_val.flow_from_directory(
    batch_size=TAMANO_LOTE,
    directory=datos_validacion,
    color_mode='grayscale',
    target_size=(TAMANO_IMG, TAMANO_IMG),
    class_mode='sparse'
)

# Modelo con L2 Regularization
modelo = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001), input_shape=(TAMANO_IMG,
TAMANO_IMG, 1)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(128, (3,3), activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(256, (3,3), activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.001)),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Compilación
modelo.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=['accuracy'])

# Callback para early stopping
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=10, restore_best_weights=True)

# Entrenamiento del modelo
print("Entrenando modelo...")
epocas = 100
history = modelo.fit(
    data_gen_entrenamiento,
    steps_per_epoch=int(np.ceil(total_entrenamiento / float(TAMANO_LOTE))),
    epochs=epocas,
    validation_data=data_gen_validacion,
    validation_steps=int(np.ceil(total_val / float(TAMANO_LOTE))),
```

```
    callbacks=[early_stopping, reduce_lr]
)

print("Modelo entrenado!")

# Guardar el modelo
modelo.save('modelo_motores_AD.h5')
print("Modelo guardado!")

# Visualización de accuracy y pérdida
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Código para el cálculo de métricas

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import precision_score, recall_score, f1_score,
accuracy_score, confusion_matrix

# Rutas a las carpetas de imágenes de prueba
test_dir = r'C:\Users\usuario\Desktop\TFM\Imagenes\test\test'

# Tamaño de lote y tamaño de imagen
TAMANO_LOTE = 100
TAMANO_IMG = 100

# Generador de imágenes para modelo sin aumento de datos
test_datagen_no_aug = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2],
    fill_mode='nearest'
)

test_generator_no_aug = test_datagen_no_aug.flow_from_directory(
    directory=test_dir,
```

```
    target_size=(TAMANO_IMG, TAMANO_IMG),
    batch_size=TAMANO_LOTE,
    color_mode='grayscale',
    class_mode='sparse',
    shuffle=False
)

# Generador de imágenes para modelos con aumento de datos
test_datagen_aug = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    brightness_range=[0.8, 1.2],
    fill_mode='nearest'
)

test_generator_aug = test_datagen_aug.flow_from_directory(
    directory=test_dir,
    target_size=(200, 200),
    batch_size=32,
    color_mode='grayscale',
    class_mode='sparse',
    shuffle=False
)

# Cargar los modelos entrenados
modelo_no_aug = tf.keras.models.load_model('modelo_motores_bn.h5')
modelo_aug = tf.keras.models.load_model('modelo_motores_AD.h5')

# Evaluar y obtener métricas
def evaluar_modelo(modelo, generator):
    y_pred_probs = modelo.predict(generator)
    y_pred = np.argmax(y_pred_probs, axis=1)
    y_true = generator.classes
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred, average='weighted')
    recall = recall_score(y_true, y_pred, average='weighted')
    f1 = f1_score(y_true, y_pred, average='weighted')
    conf_matrix = confusion_matrix(y_true, y_pred, labels=[0, 1, 2])
    return accuracy, precision, recall, f1, conf_matrix

# Evaluar modelos
accuracy_no_aug, precision_no_aug, recall_no_aug, f1_no_aug,
conf_matrix_no_aug = evaluar_modelo(modelo_no_aug, test_generator_no_aug)
accuracy_aug, precision_aug, recall_aug, f1_aug, conf_matrix_aug =
evaluar_modelo(modelo_aug, test_generator_aug)

# Imprimir resultados
print("Resultados del Modelo sin Aumento de Datos:")
print(f"Accuracy: {accuracy_no_aug}")
print(f"Precision: {precision_no_aug}")
print(f"Recall: {recall_no_aug}")
```

```
print(f"F1 Score: {f1_no_aug}")
print(f"Matriz de confusión:\n{conf_matrix_no_aug}")

print("\nResultados del Modelo con Aumento de Datos:")
print(f"Accuracy: {accuracy_aug}")
print(f"Precision: {precision_aug}")
print(f"Recall: {recall_aug}")
print(f"F1 Score: {f1_aug}")
print(f"Matriz de confusión:\n{conf_matrix_aug}")
```

Capítulo 8. BIBLIOGRAFÍA

[1] J. del C. Peralta Abarca, B. Martínez Bahena, y J. Enríquez Urbano, «Industria 4.0», *Inventio, la génesis de la cultura universitaria en Morelos, ISSN-e 2448-9026, Vol. 16, Nº. 39, 2020*, vol. 16, n.º 39, p. 4, 2020, doi:

[2] R. Mababu Mukiur, «Análisis de las competencias claves para la industria 4.0: Las competencias para la Industria 4.0.», *TECHNO REVIEW: International Technology, Science and Society Review / Revista Internacional de Tecnología, Ciencia y Sociedad, ISSN-e 2695-9933, Vol. 11, Nº. 2, 4, 2022*, vol. 11, n.º 2, p. 2, 2022, Accedido: 1 de noviembre de 2023. [En línea]. Disponible en: https://journals.eagora.org/revTECHNO/article/view/4392

[3] J. Rubio Aparicio, «Contribución al desarrollo de sistemas de telelectura inteligente con IOT», sep. 2019, doi: 10.31428/10317/8366.

[4] D. Pérez y R. H. Risco Ramos, «Implementación de Lora y Lorawan como escenario futuro de la industria 4.0 en el sector agroindustrial peruano», Revista Campus, ISSN-e 1812-6049, Vol. 25, No. 29, 2020 (Ejemplar dedicado a: Campus XXIX), págs. 133-147, vol. 25, n.o 29, pp. 133-147, 2020, Accedido: 1 de noviembre de 2023. [En línea]. Disponible en: https://www.aulavirtualusmp.pe/ojs/index.php/rc/article/view/1829

[5] M. Picquart y I. Carrasco Morales, «De la temperatura y su medición», Latin-American Journal of Physics Education, ISSN-e 1870-9095, Vol. 11, No. 1, 2017, vol. 11, n.o 1, p. 10, 2017, Accedido: 1 de noviembre de 2023. [En línea]. Disponible en: https://dialnet.unirioja.es/servlet/articulo?codigo=6019786&info=resumen&idioma=ENG

[6] R. Lizán Ortiz, V. Miquel, y R. Peñarrocha, «Dispositivo de monitorización de temperatura», oct. 2018, Accedido: 1 de noviembre de 2023. [En línea]. Disponible en: https://riunet.upv.es/handle/10251/109362

[7] L. M. Serrano Malagón y A. M. Núñez Campo, «ESTADO DEL ARTE DE LA TERMOGRAFÍA INFRARROJA COMO HERRAMIENTA EN LOS PROCESOS INDUSTRIALES», 2011. Accedido: 7 de noviembre de 2023. [En línea]. Disponible en: https://repository.upb.edu.co/bitstream/handle/20.500.11912/1644/digital_21065.pdf

[8] «Tipos de SENSORES de Temperatura y sus diferencias». [En línea]. Disponible en: https://srcsl.com/tipos-sensores-temperatura/

[9] M. Cámara, «Sensor de temperatura por fibra óptica». Accedido: 7 de noviembre de 2023. [En línea]. Disponible en: https://www.fibraopticahoy.com/sensor-temperatura-fibra-optica/

[10] «MAX31855K Thermocouple Breakout Hookup Guide - SparkFun Learn». [En línea]. Disponible en: https://learn.sparkfun.com/tutorials/max31855k-thermocouple-breakout-hookup-guide?_gl=1*1phs3z1*_ga*MjAyMDE4MjcwMy4xNjk4NzY5MjUy*_ga_T369JS7J9N*MTY5ODc2OTI1Mi4xLjEuMTY5ODc3MDgyOS41OS4wLjA.&_ga=2.268980721.101923856.1698769253-2020182703.1698769252#reading-the-temperature

[11] «SparkFun Inventor's Kit Experiment Guide - v4.1 - SparkFun Learn». [En línea]. Disponible en: https://learn.sparkfun.com/tutorials/sparkfun-inventors-kit-experiment-guide---v41/circuit-4b-temperature-sensor

[12] «T-Lite». [En línea]. Disponible en: https://docs.m5stack.com/en/app/T-Lite

[13] J. Cui y J. Wu, *2015 IEEE 12th International Conference on Electronic Measurement & Instruments (ICEMI): Qingdao, China, July 16-18, 2015*. 2015.

[14] M. Du, *«Economic Forecast Model and Development Path Analysis Based on BP and RBF Neural Network,» 2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT), Bhopal, India, 2023, pp. 619-624, doi: 10.1109/CSNT57126.2023.10134678.* 2023.

[15] C. Pappas et al., «Programmable Tanh-, ELU-, Sigmoid-, and Sin-Based Nonlinear Activation Functions for Neuromorphic Photonics», IEEE Journal of Selected Topics in Quantum Electronics, vol. 29, n.o 6, nov. 2023, doi: 10.1109/JSTQE.2023.3277118.

[16] M. S. Amin, L. Anselma, y A. Mazzei, «The Role of Activation Function in Neural NER for a Large Semantically Annotated Corpus», Institute of Electrical and Electronics Engineers (IEEE), ene. 2023. doi: 10.1109/etecte55893.2022.10007317.

[17] D. Dai, «An Introduction of CNN: Models and Training on Neural Network Models», en Proceedings - 2021 International Conference on Big Data, Artificial Intelligence and Risk Management, ICBAR 2021, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 135-138. doi: 10.1109/ICBAR55169.2021.00037.

[18] Z. Li, F. Liu, W. Yang, S. Peng, y J. Zhou, «A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects», IEEE Trans Neural Netw Learn Syst, vol. 33, n.o 12, pp. 6999-7019, dic. 2022, doi: 10.1109/TNNLS.2021.3084827.

[19] E. Dandil y M. S. Yildirim, «Automatic Segmentation of COVID-19 Infection on Lung CT Scans using Mask R-CNN», en HORA 2022 - 4th International Congress on Human-Computer Interaction, Optimization and Robotic Applications, Proceedings, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/HORA55278.2022.9800029.

[20] A. Kovacs, B. Bogdandy, y Z. Toth, «Predict Stock Market Prices with Recurrent Neural Networks using NASDAQ Data Stream», en SACI 2021 - IEEE 15th International Symposium on Applied Computational Intelligence and Informatics, Proceedings, Institute of Electrical and Electronics Engineers Inc., may 2021, pp. 449-454. doi: 10.1109/SACI51354.2021.9465634.

[21] K. S. Gill, V. Anand, R. Chauhan, A. Choudhary, y R. Gupta, «CNN, LSTM, and Bi-LSTM based Self-Attention Model Classification for User Review Sentiment Analysis», en 2023 3rd International Conference on Smart Generation Computing, Communication and Networking, SMART GENCON 2023, Institute of Electrical and Electronics Engineers Inc., 2023. doi: 10.1109/SMARTGENCON60755.2023.10442498.

[22] Prabhat, Nishant, y D. Kumar Vishwakarma, Proceedings of the International Conference on Intelligent Computing and Control Systems (ICICCS 2020) : 13-15 May, 2020. 2020.

[23] P. Cobelli, S. Nesmachnow, y J. Toutouh, «A comparison of Generative Adversarial Networks for image super-resolution», en Proceedings - 2022 IEEE Latin American Conference on Computational Intelligence, LA-CCI 2022, Institute of Electrical and Electronics Engineers Inc., 2022. doi: 10.1109/LA-CCI54402.2022.9981850.

[24] S. Kumar y S. Dhawan, A Detailed Study on Generative Adversarial Networks. 2020.

[25] J. Liu, C. Wang, H. Su, B. Du, y D. Tao, «Multistage GAN for Fabric Defect Detection», IEEE Transactions on Image Processing, vol. 29, pp. 3388-3400, 2020, doi: 10.1109/TIP.2019.2959741.

[26] S. Karthika y M. Durgadevi, «Generative Adversarial Network (GAN): A general review on different variants of GAN and applications», en Proceedings of the 6th International Conference on Communication and Electronics Systems, ICCES 2021, Institute of Electrical and Electronics Engineers Inc., jul. 2021. doi: 10.1109/ICCES51350.2021.9489160.

[27] «Web oficial de TensorFlow».

[28] «Objetivos de Desarrollo Sostenible | Programa De Las Naciones Unidas Para El Desarrollo». [En línea]. Disponible en: https://www.undp.org/es/sustainable-development-goals