Universidad de Valladolid

Máster en Matemáticas

Uso de técnicas avanzadas de análisis de datos para detección y clasificación de fallos en motores de inducción

Trabajo de Fin de Máster

Autor

Alejandro Barón García

Tutor

Miguel Alejandro Fernández Temprano

Resumen

El mantenimiento predictivo (el conjunto de técnicas que permite detectar posibles averías antes de que sucedan) resulta de gran interés en la industria. El poder detectar con antelación problemas reduce costos de mantenimiento, funcionamiento y producción, siendo de especial relevancia cuando permite realizar una monitorización del estado de deterioro de los distintos aparatos sin realizar una inspección invasiva.

Ese es precisamente el caso del *Motor Current Signature Analysis* en el que, midiendo la corriente de alimentación de motores de inducción (presentes en múltiples aplicaciones industriales), podemos ver el comportamiento de los armónicos inducidos sobre esta debido a inestabilidades producidas por deterioros. En función de estos armónicos y de su intensidad, se puede inferir el estado de deterioro en el que se encuentra el motor.

Hasta ahora, esta predicción se realizaba en base a medidas tabulares, es decir, información estructurada en forma de tablas mediante una representación matemática de la realidad de la observación a estudiar. Sin embargo, los avances en el campo de la inteligencia artificial y, en concreto, el *Deep Learning*, permiten emplear otro tipo de información no estructurada, principalmente mediante técnicas de *Computer Vision* (imágenes) y de *Natural Language Processing* (texto), siendo la primera de posible utilidad en el problema que abarca esta memoria. Mediante una representación no estructurada tiempo-frecuencia de la corriente de alimentación, llamada espectrograma, se puede aprovechar la información que antes se alimentaba de forma tabular, permitiendo a mayores a nuestro algoritmo de clasificación inferir el estado de deterioro no solo a partir de valores concretos de ese espectrograma si no a partir de relaciones espaciales entre estos.

Esta memoria plasma el trabajo de la clasificación de motores de inducción en estados de deterioro mediante técnicas de *Deep Learning*. En concreto, se usarán redes neuronales convolucionales junto a procedimientos de *Transfer Learning* que nos permitirá trabajar con modelos potentes aún teniendo poca muestra (dado que los algoritmos de *Deep Learning* requieren de muchos datos para ser usados), siendo ambas técnicas parte del estado del arte. Es un trabajo puntero que no solo requiere de técnicas de inteligencia artificial de las más avanzadas si no que también precisa de programación avanzada para su desarrollo.

Abstract

Predictive maintenance (the set of techniques to detect potential failures before they occur) is of great interest in industry. Being able to detect problems in advance reduces maintenance, operating and production costs, especially as it allows monitoring of the state of deterioration of the various devices without invasive inspection.

Among these non-invasive techniques is the *Motor Current Signature Analysis* in which, by measuring the power supply current of induction motors (present in many industrial applications), we can see the behaviour of the harmonics induced on it due to instabilities caused by deterioration. Depending on these harmonics and their intensity, the state of deterioration of the motor can be inferred.

Until now, this prediction was made on the basis of tabular measurements, i.e. information structured in the form of tables, a mathematical representation of the reality of the observation to be studied. However, advances in the field of artificial intelligence and, specifically, Deep Learning, make it possible to use other types of unstructured information, mainly by means of *Computer Vision Techniques* (images) and *Natural Language Processing* (text), the former being of possible use in the problem covered by this report. Using an unstructured time-frequency representation of the feed stream, the so called spectrogram, we can take advantage of information that was previously fed in tabular form, plus allowing our classification algorithm to infer the state of deterioration not only from specific values in that spectrogram but also from spatial relationships between them.

This report presents the work on the classification of induction motors in states of deterioration using Deep Learning techniques. Specifically, Convolutional Neural Networks will be used together with Transfer Learning procedures that will allow us to work with powerful models even with a small sample (despite the fact that Deep Learning algorithms require a lot of data to be used), both techniques being part of the state of the art. It is a cutting-edge work that not only requires some of the most advanced artificial intelligence techniques but also advanced programming for its development.

Agradecimientos

A mis padres y a mi tutor Miguel Alejandro.

Índice general

1. Introducción

1.1. Descripción del problema

En entornos de producción industrial, los motores representan el corazón de muchos procesos. Es por eso que evitar fallos en las máquinas e instrumentos antes de que sucedan es crítico. Si somos capaces de predecir cuándo va a fallar un motor con antelación en base a medidas externas que reflejen el estado de deterioro, podemos ahorrar costes de mantenimiento, producción, etc. Esta familia de procedimientos se denomina **mantenimiento predictivo**.

Especialmente interesante es el estudio de estas técnicas aplicadas a motores eléctricos de inducción. De hecho, los motores de inducción representan entre un 80 % y un 85 % del consumo energético de la industria [1], por lo cual emplear este tipo de técnicas representa un gran interés. Dos elementos claves a tener en cuenta en este proceso son el **inversor** y el **nivel de carga**, cuya influencia se vio demostrada a la hora de entrenar un clasificador en [2].

El uso de metodología estadística, matemática y computacional representa el estado del arte del mantenimiento predictivo. Hasta ahora, se han empleado técnicas asociadas a información tabular o estructurada, véanse por ejemplo [1] donde se emplean técnicas estadísticas o [2] donde se emplea Machine Learning . Sin embargo, en los últimos años, la explosión computacional ha propiciado la irrupción de nuevas técnicas basadas en Deep Learning. A pesar de que sigue habiendo camino por recorrer en el campo de los datos tabulares, surgiendo alternativas que palian los problemas de las redes neuronales en este tipo de datos como TabNet [3], el Deep Learning aprovecha toda su potencia y sigue ostentando la hegemonía en el uso de información no estructurada, como el caso de la visión computacional o el procesado del lenguaje natural.

En este Trabajo de Fin de Máster (TFM) se aúnan ambos caminos mediante el uso de información no estructurada para mantenimiento predictivo. La fuente de información empleada es el **espectrograma**, una representación en tiempo-frecuencia de la intensidad de las componentes espectrales de la onda a estudiar, en este caso, la corriente de alimentación del motor, sobre la cual, debido a las vibraciones a causa del deterioro, se inducen perturbaciones que excitan determinadas frecuencias. En trabajos anteriores como [2] se emplearon concretamente la evolución temporal de dos armónicos característicos que nacen debido a roturas del rotor [4]: el *Lower Sideband Harmonic* o LSH y el *Upper Sideband Harmonic* o USH.

El restringirnos a la observación de armónicos o frecuencias concretas supone la pérdida de valiosa información que podemos estar obviando al tabular los datos. Además, se pierden posibles relaciones espaciales que pueden influir en la respuesta (puede influir no solo la intensidad de dos frecuencias concretas, si no la intensidad de dos de ellas en función de su localización espacial, siendo este tipo de relaciones difíciles de plasmar y ser inferidas mediante datos tabulares).

Por ello, en este trabajo se expone una metodología novedosa basada en el uso del espectrograma como información no estructurada (en este caso, como una imagen) para mantenimiento predictivo. El uso de espectrogramas se empleó en éxito en trabajos como YamNet [5] donde se emplea el *Mel Spectrogram* para clasificación de fonemas, siendo esta una de las aproximaciones más populares.

1.2. Objetivos del proyecto

En esta memoria de TFM se presenta un trabajo de investigación que pretende mejorar los resultados obtenidos hasta la fecha en la detección y clasificación de fallos en motores de inducción empleando técnicas novedosas del estado del arte de la Inteligencia Artificial.

En concreto, se emplearán técnicas de *Computer Vision* basadas en Redes Neuronales Convolucionales para la clasificación de espectrogramas. Estamos ante un reto ya que el tamaño muestral que se manejará es escaso para los volúmenes manejados habitualmente en el ámbito del Deep Learning. Sin embargo, mediante el uso de técnicas avanzadas como el *Transfer Learning* se puede realizar un baipás a este problema y solventarlo.

Los objetivos de este proyecto son los siguientes:

- **Uso de información no estructurada** en forma de espectrograma, para tratar de *exprimir* al máximo la información disponible.
- **Uso de técnicas punteras de Inteligencia Artificial** de cara a traer a este campo algoritmos del estado del arte que triunfan en otros campos de aplicación.
- **Mejora de los resultados obtenidos hasta la fecha** gracias a la combinación de los dos puntos anteriores.
- Verificación de los conceptos aprendidos por el clasificador mediante **técnicas de interpretabilidad**, así como el estudio de qué regiones del espectrograma influyen en la clasificación. De este modo, podremos desencriptar los modelos de caja negra.
- Estudio de similitudes y diferencias en las combinaciones de **inversor** y **nivel de carga** en el desempeño en la clasificación de los modelos entrenados, de cara a buscar, para cada inversor, el mejor nivel de carga asociado a la hora de predecir.
- Estudio de la posible **mejora al incluir información del estado estacionario**: en esta memoria se trabajará en su mayoría con información del estado transitorio, presentando esto un mayor reto ya que la onda principal no es estacionaria y por tanto sus características varían con el tiempo en esta franja. Se tratará de comprobar si el uso de la información del estado transitorio es suficiente o el incluir el estado estacionario mejoraría los resultados.

Los datos empleados en este TFM fueron proporcionados por el Departamento de Ingeniería Eléctrica de la Universidad de Valladolid

1.3. Estructura del documento

Esta memoria se compone de los siguientes capítulos:

- **Metodología**: en el que se exponen los aspectos y fundamentos teóricos de las técnicas empleadas en este TFM.
- **Descripción y procesado de los datos**: en el que se describen los datos, su estructura y balance por clases.
- **Ajuste de Modelos y estudio de resultados**: en el que se muestra el procedimiento de ajuste de hiperparámetros así como el comentario sobre resultados obtenidos tanto numéricos como de interpretación.

- **Conclusiones y trabajo futuro**: donde se finaliza el proyecto con reflexiones sobre los contenidos de la memoria y propuestas de trabajo de cara a explorar nuevas vías de investigación.

1.4. Relación con las asignaturas del Máster

- **Aprendizaje Automático**, **Seminario de Modelización Estadística**, **Técnicas escalables de análisis de datos en entornos *Big Data*: Clasificadores** y **Técnicas escalables de análisis de datos en entornos *Big Data*: Regresión y Descubrimiento de Conocimiento** en las que se expone la metodología estándar de Técnicas de Aprendizaje Automático empleadas en essta memoria.

- **Análisis de Señales Biomédicas** en la que se exponen conceptos tanto del estudio de señales basado en espectrogramas como en el análisis estadístico de este tipo de información.

- **Modelos de Optimización** en la que se exponen conceptos en cuanto a optimización de funciones objetivo.

- **Programación y Análisis de Datos con R** en la que se exponen buenas prácticas a la hora de programar proyectos de análisis de datos.

1.4.1. Ampliación de Materia

- **Uso de técnicas de Deep Learning**: durante el máster, se exponen conceptos comunes a toda la teoría del análisis de datos aplicados a distintos tipos de modelos. Este TFM se vertebra en torno a las técnicas de Deep Learning, la rama de la Inteligencia Artificial que está proporcionando resultados más potentes a día de hoy.

- **Uso de métodos de interpretabilidad**: la mayor crítica hacia los modelos de Deep Learning es su falta de interpretabilidad, acusados de ser una caja negra. Empiezan a surgir técnicas como LIME, SHAP o Grad-CAM que permiten *abrir* esta caja negra y ver qué conceptos está aprendiendo el clasificador.

- **Técnicas de programación avanzada**: En este TFM, no solo se lleva un trabajo de investigación basado en técnicas del estado del arte, si no que además se programa la solución desde cero en uno de los lenguajes de programación más

punteros como es Python y la librería Tensorflow para técnicas de aprendizaje profundo.

2. Metodología

2.1. Trabajo Previo

El desarrollo de este proyecto se enmarca en el ámbito de los trabajos de mantenimiento predictivo desarrollados por el Grupo de Investigación Reconocido (GIR) Inferencia con Restricciones en colaboración con el GIR Análisis y Diagnóstico de Instalaciones y Redes Eléctricas. La intención de estos trabajos es la de crear sistemas de mantenimiento predictivo basados en métodos estadísticos de clasificación para la predicción de fallos en motores.

Hasta ahora, se han llevado a cabo con éxito diversas aproximaciones como es el uso de técnicas *Boosting* [2] o de métodos Lasso [1]. Sin embargo, existen vías aún por explorar como es el uso de técnicas Deep Learning que hasta hace poco no eran accesibles por el público general, debido a la complejidad de la programación (surgiendo frameworks avanzados que permiten le empleo de estas) y a la democratización de la capacidad computacional.

2.2. Uso de información no estructurada

De cara a realizar mantenimiento predictivo, hasta ahora se ha trabajado con datos tabulares (datos estructurados en matrices con observaciones por filas y variables por columnas). El uso de datos tabulares trae ventajas como el ser una reducción de la complejidad del problema al representar, para cada observación o, las realidad de esa observación como un vector de datos reales $o \in \mathbb{R}^p$, siendo p el número de variables. Esta *matematización* de la realidad trae ventajas como son facilidad de cálculo y de algoritmos empleados.

De este modo, se pueden aplicar técnicas potentes de clasificación supervisada basada en datos tabulares, siendo estos la entrada que esperan la mayoría de algoritmos de *Machine Learning* (ML). Sin embargo, en los últimos años ha habido un auge en el uso de información no estructurada (i.e. imágenes o texto) como entrada para algoritmos de ML. En este caso, se deja al propio algoritmo que aprenda una codificación o representación interna de la observación a estudiar. De este modo, seguimos traduciendo la realidad compleja de la observación (una imagen, una señal de audio o una sentencia de un texto) a una codificación de un vector real. No obstante, a diferencia de cuando usamos datos tabulares donde nosotros imponemos la codificación en forma de variables, cuando usamos información no estructurada es el propio algoritmo el que aprende además de la tarea de interés (regresión, clasificación...) la codificación de la propia observación.

Ventajas del uso de información no estructurada

- Mayor flexibilidad a la hora de representar la información
- Mayor cantidad de información como entrada
- Facilidad de la captura y procesado de datos de cara al desarrollo de soluciones

Desventajas del uso de información no estructurada

- Necesidad de muchos datos para aprender la realidad del problema
- Algoritmos más complejos
- Dificultad de interpretación de los resultados

En este trabajo de fin de grado se tomará una aproximación Deep Learning para clasificar el estado de deterioro de un motor en función de sus espectrograma, estudiado como una imagen.

2.2.1. Espectrograma

En trabajos anteriores se usaron datos tabulares registrando, en cada instante de tiempo las medidas de los armónicos de la onda principal, como en [2, 1] utilizándose en [2] las de los armónicos LSH y USH en el estado transitorio del motor.

Para este trabajo, se empleará el espectrograma como representación de la observación, que contiene toda la información utilizada en las aproximaciones anteriores, de forma no estructurada, en una sola imagen.

El espectrograma consiste en una representación visual en tiempo-frecuencia de la onda a estudiar. En el eje de ordenadas, se representan las frecuencias y en el de abscisas, el tiempo. El color de cada par (t, f) representa la intensidad de la componente en frecuencia de la onda f en el instante t. De este modo, podemos ver cómo evolucionan las componentes en frecuencia a lo largo del tiempo.

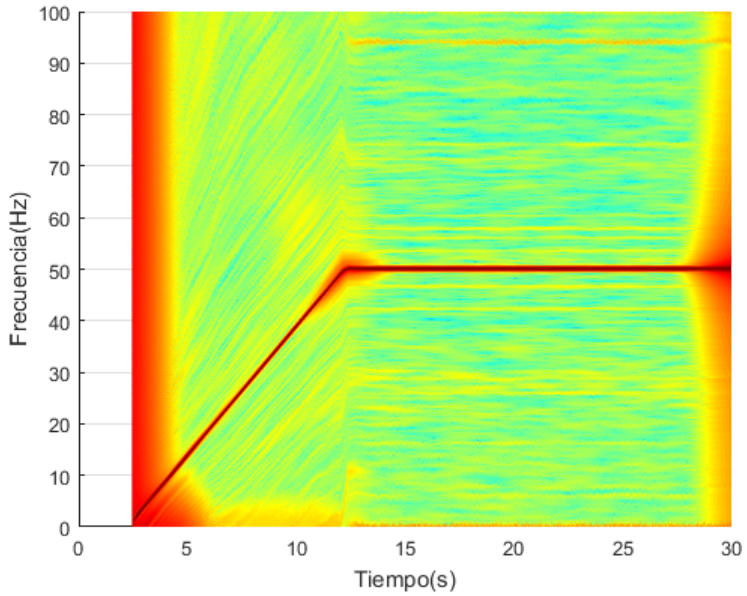

Figura 2.1: Ejemplo de espectrograma en el problema del mantenimiento predictivo de motores

Para obtener el espectrograma, se debe realizar una extracción (mediante una **transformada**) de los armónicos que componen la onda (donde registramos la corriente de alimentación del motor). La transformada considerada como estándar frente al que comparar en el dominio de aplicación que nos concierne es la transformada Transformada de Fourier de Tiempo Reducido (STFT de sus siglas en inglés) [4]. Sin embargo, esta transformada posee una resolución tiempo-frecuencia fija, por lo que surgen otras alternativas basadas en Wavelets, como la desarrollada por el GIR Análisis y Diagnóstico de Instalaciones y Redes Eléctricas, la transformada Dragón [6] con el objetivo de solventar los problemas que padece la STFT.

La motivación del uso de espectrogramas viene derivada del éxito de aplicación en otros

campos como el procesado de señales biomédicas [7], en la clasificación de audio [8], sistemas *anti-malware* [9] o en mantenimiento predictivo [10, 11]. En [12], emplearon redes convolucionales 1D en el ámbito del *Motor Current Signature Analyisis*. Existen algunas aproximaciones basadas en las técnicas que aquí se exponen como [13], sin embargo, en este caso la metodología usada no utiliza la triple partición train-val-test que se requiere en el ámbito del Deep Learning (con un sospechoso acierto del 100 %, amén de una clasificación de deterioros más sencilla que la que nosotros usamos) y los espectrogramas empleados no se asemejan a los obtenidos en nuestro trabajo dado que utilizan la medida de la corriente directamente a través de la red (que es más fácil de clasificar), por lo que el dominio de aplicación se considera distinto. También existen otras aproximaciones basadas en algunas de las técnicas aquí expuestas como [14, 15, 16, 17], principalmente basadas en sensores de vibración y no de corriente como los que aquí usamos.
En este trabajo, se aúnan todas las técnicas usadas en estos trabajos (todos ellos de menos de dos años de antigüedad) en un solo proyecto puntero mediante una metodología más robusta.

2.3. Deep Learning

Dentro de los modelos de ML, encontramos un subconjunto denominado Deep Learning. La principal característica de esta familia de técnicas es la complejidad de los modelos frente a los modelos de Machine Learning o Statistical Learning.

Habitualmente, se considera como primer modelo de *red neuronal artificial* (en adelante ANN de sus siglas en inglés Artificial Neural Network) el propuesto por Warren S. McCulloch & Walter Pitts en 1943 [18]. Más adelante, Alexey Ivakhnenko y Lapa [19] propusieron en 1967 el primer modelo de ANN multicapa de tipo *feedforward*.

En cualquier caso, no fue hasta la pasada década cuando el Deep Learning se hizo hueco en el estado del arte, ostentando la hegemonía en determinadas aplicaciones. Parte de este éxito surgió en 1986, cuando David E. Rumelhart, Geoffrey E. Hinton y Ronald J. Williams [20] presentaron la aplicación del algoritmo de *backpropagation* aplicado a ANNs, lo que permitió un ajuste basado en muestras de los pesos de la ANN de forma rápida y sencilla. Sumado a la gran potencia de cálculo que consiguieron los ordenadores en los primeros años del siglo XXI, permitieron el auge de las *Deep Neural Networks* o DNNs.

Uno de los grandes hitos de la pasada década que causó la explosión del interés por el

Deep Learning fue *AlexNet* [21]. En 2012, Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton entrenaron una red neuronal con cómputos basados en GPUs, incluyendo técnicas novedosas como el uso de funciones de activación ReLUs en vez de Sigmoides. *AlexNet* consiguió unas tasas de error del 37.5 % y del 17.0 % para el problema top-1 y top-5 respectivamente, mientras que las tasas de error de los algoritmos imperantes fueron del 47.1 % y del 28.2 % para sendos problemas, motivando así el uso de redes neuronales, en este caso redes neuronales convolucionales, que será el algoritmo que vertebre este trabajo.

2.4. Perceptrón Multicapa

En primer lugar, conviene exponer la primera arquitectura profunda que fue el Perceptrón Multicapa (MLP de sus siglas en inglés). La Figura 2.2 muestra un ejemplo de un MLP.

La topología de un MLP viene definida por el número de capas y el número de neuronas en cada capa, siendo cada neurona (también conocida como perceptrón simple) una unidad de cálculo sencillo. Concatenando la salida de cada neurona de la capa anterior con la entrada de cada neurona de la siguiente capa, construimos el MLP.

Estas redes, habitualmente conocidas por *fully-connected feed-forward networks* (en cuanto a que todas las conexiones entre neuronas de capas inmediatamente consecutivas están presentes) pueden ser utilizadas por sí mismas o a la salida de otro tipo de redes, como comentaremos más adelante. Conviene pararnos a estudiar qué sucede dentro de cada neurona o perceptrón simple, como refleja la Figura 2.3.

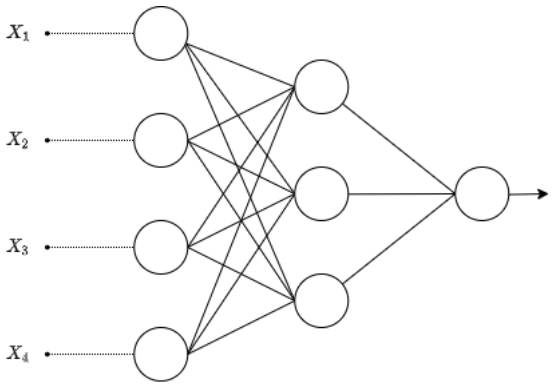

Figura 2.2: Ejemplo de MLP de 4 entradas, 2 capas y una neurona de salida

2.4.1. Perceptrón simple

Las neuronas o perceptrones simples son unidades de cálculo sencillo que permiten crear relaciones complejas entre las entradas. En cada neurona, tomadas las entradas X, se calcula una combinación lineal de X con pesos W, pudiendo sumar un término tipo intercepto o sesgo b. Sobre esta combinación lineal WX se aplica una función de activación F, proporcionando a la salida un único valor $F(WX) \in \mathbb{R}$

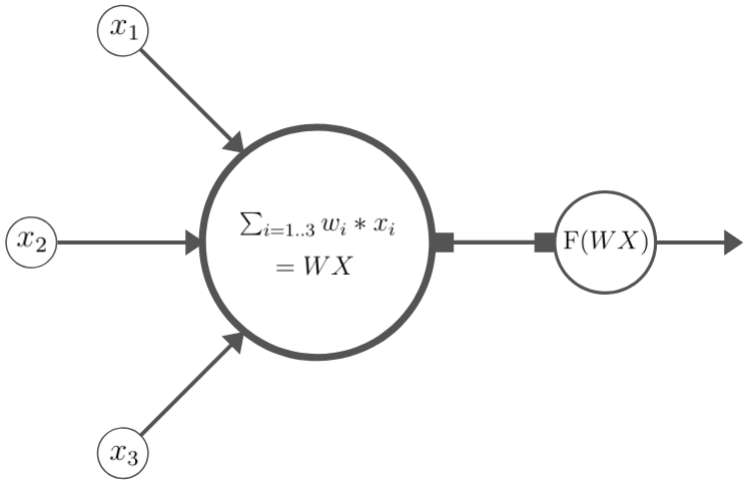

Figura 2.3: Esquema de un perceptron simple de tres entradas. Imagen de [22]

2.4.2. Funciones de activación

La función $F : \mathbb{R} \to \mathbb{R}$ que se muestra en la figura 2.3 se denomina función de activación. Originalmente, se usaba la función sigmoide pero debido al problema de los *vanishing gradients* se propusieron otras soluciones como la función de activación ReLU.

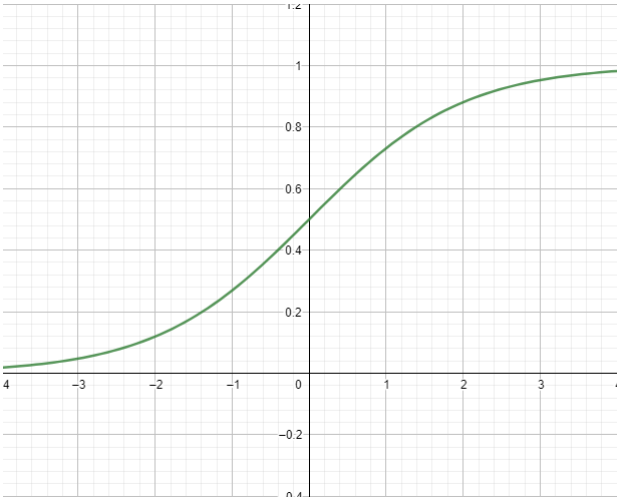

Figura 2.4: Función de Activación Sigmoide F(x) = $\frac{1}{1+e^{-x}}$

ReLU

Propuesta por Hahnloser [23], la Rectified Linear Unit (ReLU) es la función de activación $F(x) = max(x, 0)$. Dahl et al. [24] mostraron que el uso de ReLUs y Dropout mejoraban la precisión de los algoritmos frente al uso de sigmoides.

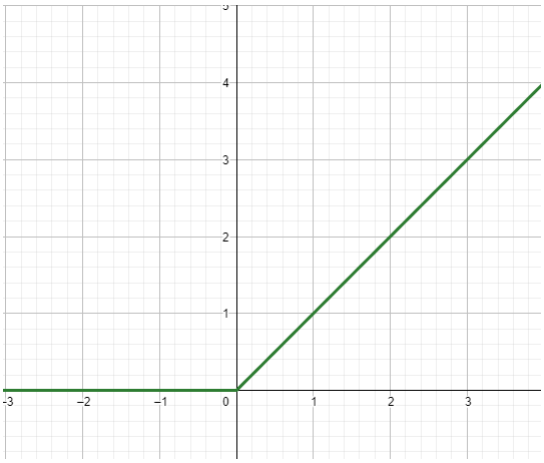

Figura 2.5: Función de Activación ReLU F(x)= max(x,0)

Softmax

La función de activación softmax permite mapear las entradas de k neuronas (siendo k el número de clases) a una probabilidad entre 0 y 1. La función softmax está presente en algoritmos clásicos como la regresión logística multiclase:

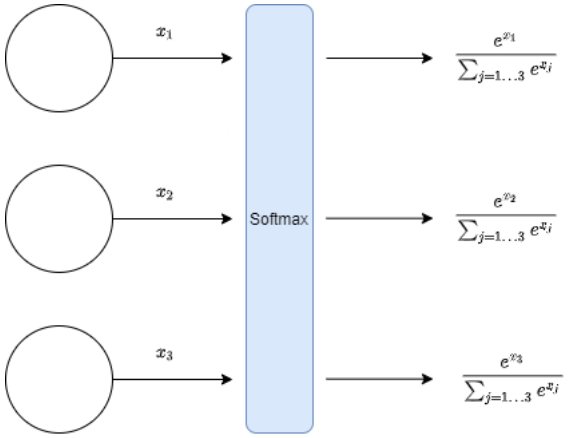

Figura 2.6: Función de Activación Softmax para tres clases

2.5. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNNs de sus siglas en inglés) son una arquitectura de ANN empleada en áreas como la visión artificial o el procesado de señales. Su arquitectura se basa en el uso de convoluciones sobre la entrada de la capa anterior, concatenándolas y aprendiendo conceptos cada vez más complejos (en el caso de las imágenes, formas o patrones)

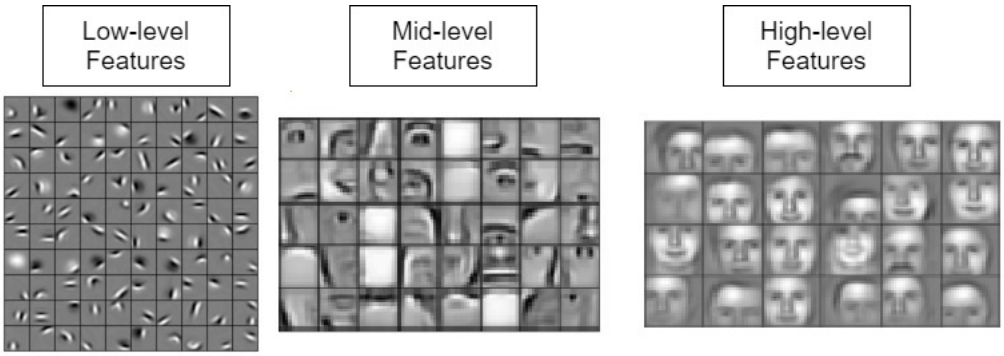

Figura 2.7: Ejemplo de aprendizaje de patrones crecientemente complejos, extraído de [25]

En el caso de las redes neuronales convolucionales, no hablamos de número de neuronas si no de número de filtros para identificar el tamaño de la capa y el tamaño de cada filtro .

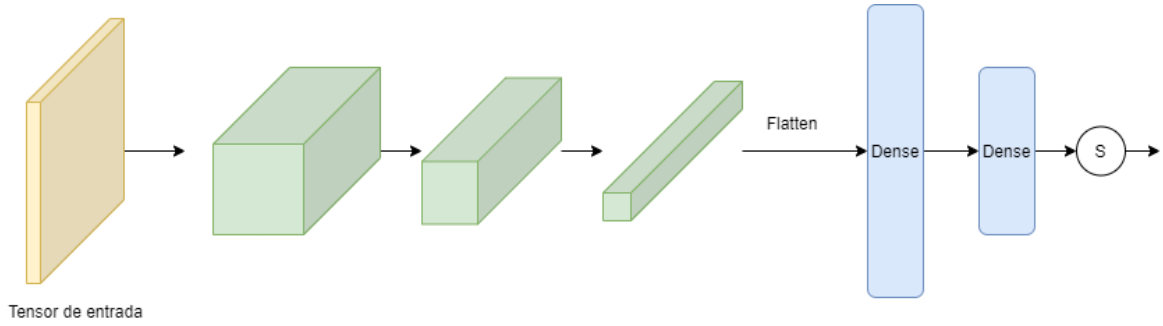

Figura 2.8: Ejemplo de CNN, cada bloque verde representa una etapa de Convolución y/o Pooling

2.5.1. Convoluciones y Pooling

En el ámbito del Deep Learning, se denomina convolución a una matriz de dimensiones $(k \times w \times d)$ (habitualmente $k = w$ en convoluciones de dos dimensiones y $k = 1$ en convoluciones de una dimensión) que se aplica sobre un tensor de entrada con d canales de información. Esta operación deriva del concepto más general de convolución muy habitual en análisis matemático y teoría de la señal.

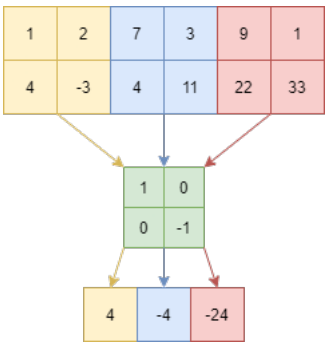

Figura 2.9: Ejemplo de aplicar un filtro en dos dimensiones, con un canal de información y con *stride=2*, siendo cada color un subtensor

La idea es extraer patrones a partir de la información contenida en el tensor de entrada (habitualmente, una imagen *alto* $\times$ *ancho* $\times$ *colores*) realizando una multiplicación elemento a elemento de la convolución sobre cada uno de los subtensores T_{ij} (marcados por el *stride* entendido como la distancia entre subtensores) de la entrada con las mismas dimensiones $(k \times w \times d)$ del filtro y sumando los elementos del tensor resultante. Siendo T nuestro tensor de entrada, F nuestro filtro o convolución, podemos expresar como sigue la operación.

$$\text{Conv2D}(T, F) = [t_{ij} |\, t_{ij} = \sum_{t_{ijk} \in T_{ij} \circ F} t_{ijk}\, , T_{ij} \in T]$$

El Pooling funciona de forma similar, pero en vez de la suma de los valores de $T_j \circ F$ (multiplicación elemento a elemento), se toma como valor otro estadístico G (máximo, promedio...) de los subtensores que marque el filtro de pooling.

$$\text{Pooling2D}(T, F) = [t_{ij} |\, t_{ij} = G(T_{ij})\, , T_{ij} \in T]$$

En el caso de las CNN, en lugar de ajustar pesos, se ajustan los valores de las convoluciones que conforman cada capa.

2.5.2. Bloques empleados en MobileNet

MobileNetV2, el modelo base utilizado del que hablaremos más adelante, introduce *bloques bottleneck* de convoluciones que siempre tienen la misma estructura:

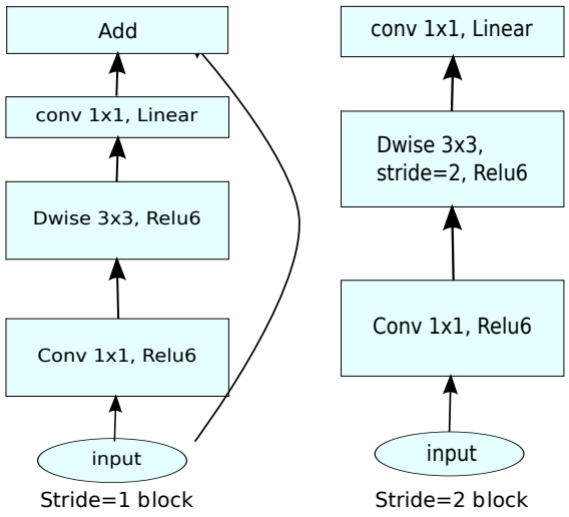

Figura 2.10: Los dos tipos de bloques *bottleneck* de MobileNetV2 en función del stride

- En primer lugar, una capa convolucional con función de activación ReLU6 (ReLU capada a un valor máximo de 6)
- En segundo lugar, una convolución Depthwise. Las convoluciones Depthwise se realizan canal a canal, en lugar de utilizar todos los canales de información del tensor de entrada.

- En tercer lugar:

  Si el stride = 1, la salida de la convolución depthwise se pasa por otra capa convolucional y finalmente, se suma esta salida con la entrada original de la capa. Esto no cambia la dimensión del tensor, filtrándolo

  Si el stride = 2 (esto reduce la dimensionalidad), la salida simplemente se pasa a través de una capa convolucional

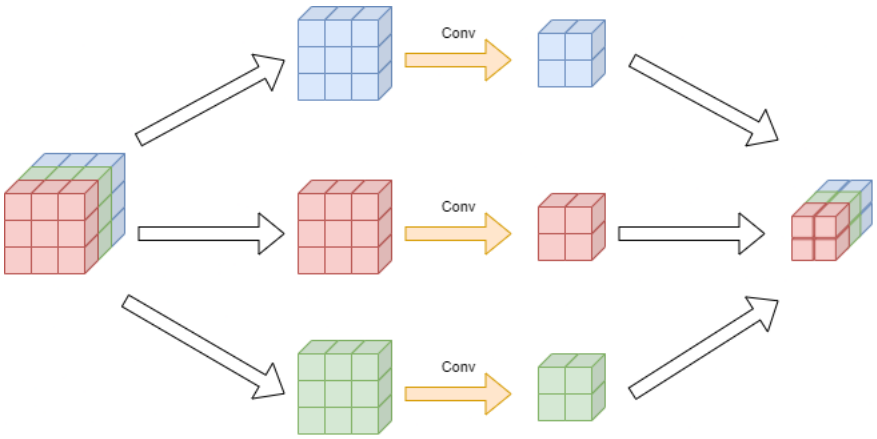

Figura 2.11: Ejemplo de Depthwise Convolution

2.6. Transfer Learning

Una técnica interesante y de gran utilidad es el Transfer Learning. Se define como Transfer Learning a las técnicas de trasvase de conocimiento de un problema a otro similar. Para facilitar el aprendizaje, se toma un algoritmo preentrenado en un dominio de aplicación fuente, a priori similar al dominio de destino [26].

La idea es la de aprovechar, de un modelo preentrenado, las primeras capas que a priori han aprendido a encontrar formas y patrones sencillos. *Congelando* estas primeras capas preentrenadas (cuya salida se puede entender como una codificación o *embedding* del tensor de entrada), podemos realizar un ajuste fino o *fine-tuning* de las capas finales. Lo ideal para el problema que en esta memoria se trata sería una red preentrenada para clasificar espectrogramas. Existen algunas opciones como Yamnet [5] popularmente empleada para clasificar sonidos basándose en el espectrograma de su señal y realizar transfer learning, que basa su arquitectura en un MobileNet. También se pueden encontrar otros modelos preentrenados en Keras Applications [27] sobre el conjunto de datos ImageNet [28].

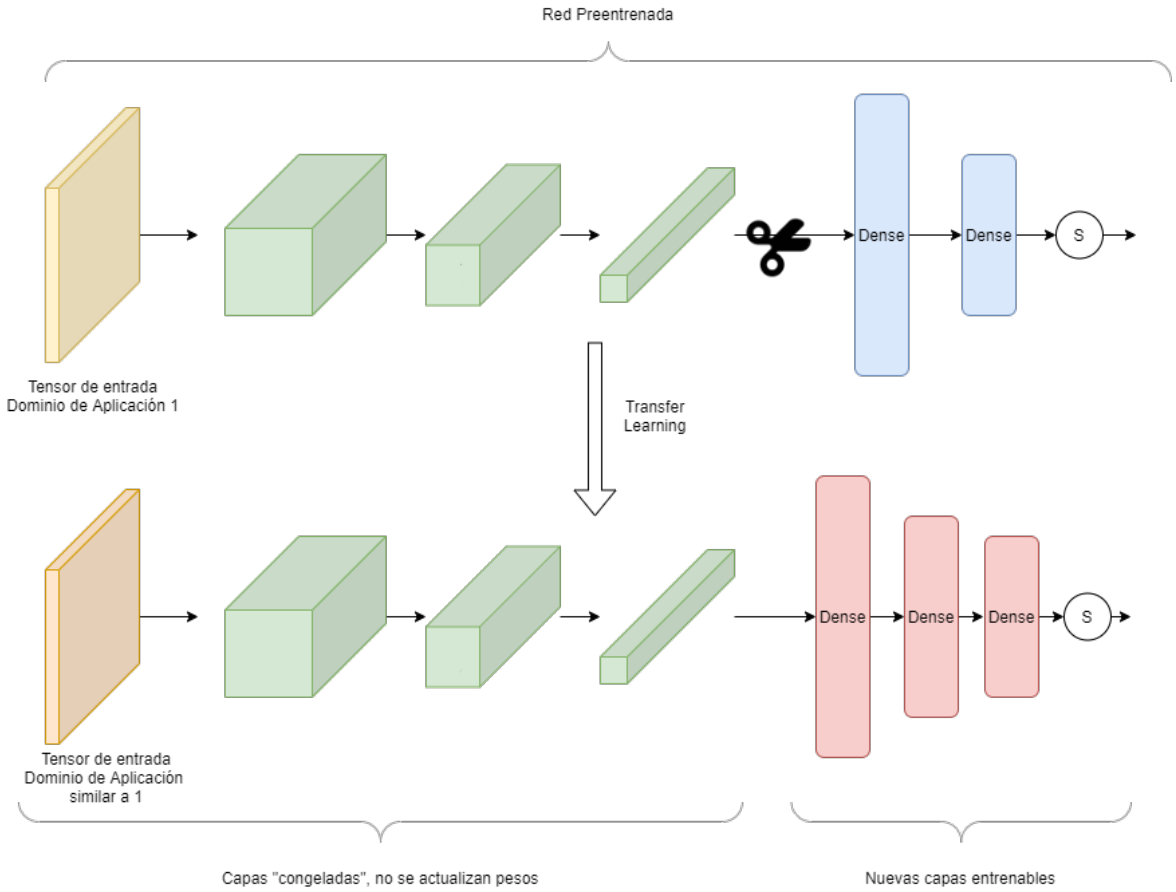

Figura 2.12: Ilustración del Transfer Learning

En este trabajo se empleará como red base la MobilenetV2 [29] disponible en Keras Applications. Los motivos de la elección fueron:

- Menor número de parámetros, lo que arroja unos menores tiempos de cómputo
- Mejores valores en las pruebas iniciales

No obstante, como se indicaba, hay alternativas que a priori pueden ser mejores como Yamnet, que está preentrenada sobre espectrogramas. Sin embargo, no se consiguieron resultados relevantes en las primeras pruebas (lo que no quiere decir que no se posible con una exploración más profunda de esta opción). Además, Yamnet ingiere tensores de espectrogramas de 1 canal (escala de grises, suficiente para representar un espectrograma), mientras que MobileNet está preparada para ingerir tensores de 3 canales (en escala de color RGB, lo cual hace que dos de los canales sean información redundante del tercero). Cabe mencionar que la mayoría de modelos están entrenados sobre

conjuntos de imágenes de naturaleza muy distinta al dominio de aplicación, como es nuestro caso con ImageNet. Sin embargo, se pueden encontrar casos de éxito como [30] en el que utilizaron un modelo ResNet para clasificar espectrogramas de señales cardíacas en distintas cardiopatías.

La descripción del modelo se incluye en la tabla 2.1

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	–	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	–	1280	1	1
$7^2 \times 1280$	avgpool 7x7	–	–	1	–
$1 \times 1 \times 1280$	conv2d 1x1	–	k	–	

Tabla 2.1: Descripción de MobileNetV2 [29], donde t es un factor de reescalado, c el tamaño de salida, n el número de veces que se repite el bloque o capa y s el stride usado

2.7. Algoritmo de Entrenamiento: Backpropagation

Esta sección se desarrolla en base a la sección 3.1.8 de [22]. La idea del ajuste de pesos mediante el algoritmo de backpropagation [20] data de 1986, surgiendo entonces una heurística para el entrenamiento basada en el gradiente de la función de pérdida. El esquema general del entrenamiento es el siguiente:

- **Paso forward**: Se alimenta a la red con las observaciones obteniendo la predicción para dichas observacies propagando sus valores hacia adelante
- **Cálculo de la función de pérdida**: Se calcula la el valor de la función de pérdida enfrentando los valores reales frente a los predichos
- **Cálculo de gradientes**: En base a la función de pérdida, calculamos su gradiente con respecto a los pesos a actualizar
- **Paso backward:** Se actualizan los pesos mediante el valor del gradiente y el hiperparámetro *learning rate*

La Figura 2.13 ilustra el algoritmo de backpropagation mediante el grafo computacional de una red neuronal de dos capas, una representación de los cálculos que se suceden en un MLP, donde $\mathcal{L}$ es la función de pérdida.

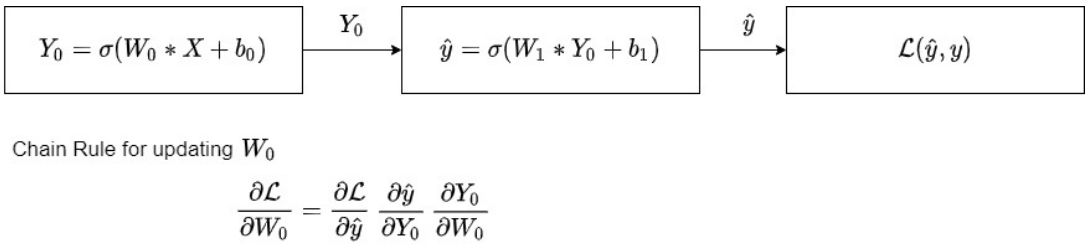

Figura 2.13: Ejemplo de [22] ilustrando la backpropagation con una ANN de dos capas

2.7.1. Ajuste de pesos

Función de pérdida

En el problema de la clasificación multiclase, la pérdida más popular y que se empleará en este trabajo es la *categorical cross-entropy*:

$$\mathcal{L}(y_i, \hat{y}_i) = -\sum_{c=0}^{\#clases} y_i^c \cdot \log \hat{y}_i^c$$

Donde $y_i^c = 1$ si la clase real de la observación y_i es c, 0 en caso contrario e $\hat{y}_i^c$ es la probabilidad estimada de pertenencia de la observación y_i a la clase c

Optimizador: Stochastic Gradient Descend

El optimizador que se empleará en este trabajo es el *Stochastic Gradient Descend*(SGD) basado en *minibatches*, subconjuntos de observaciones empleados para dar cada paso del descenso de gradiente. Siendo W^t los parámetros de nuestro modelo en el instante t, la actualización de los pesos W^{t+1} se computa como sigue:

$$W^{t+1} = W^t - \eta \frac{1}{m} \sum_{i=0}^{m} \nabla \mathcal{L}(x_i, W^t)$$

Donde η representa el hiperparámetro *learning rate* o tamaño de paso a tomar en función del gradiente. El promedio de gradientes calculados en m observaciones es la estimación del gradiente basada en las m observaciones que conforman el *minibatch*.

Momento: La introducción de momento en el proceso de entrenamiento consiste en realizar una media ponderada entre gradiente de la etapa actual y el gradiente de la etapa anterior. Sea $G^t = \frac{1}{m}\sum_{i=0}^{m} \nabla\mathcal{L}(x_i, W^n)$ el gradiente para la etapa actual actual y G^{t-1} el momento calculado en la etapa anterior. La nueva forma de actualizar los pesos se calcula como sigue:

$$G^t = \beta G^{t-1} + (1-\beta)\frac{1}{m}\sum_{i=0}^{m} \nabla\mathcal{L}(x_i, W^t)$$

$$W^{t+1} = W^t - \eta G^t$$

Donde β es un hiperparámetro que regula el peso que se le da a las etapas anteriores a la hora de estimar. El momento ayuda a la estabilidad de la convergencia de la solución.

Existen otros optimizadores como Adam que poseen un carácter adaptativo e incluyen momentos de hasta segundo orden, de cara a regular los pasos de descenso en función de lo anterior. Sin embargo, no se emplearán en este TFG ya que, aunque su convergencia es más rápida, generalmente proporcionan peores modelos a la hora de generalizar, como ilustran [31] [32].

Entrenamiento por épocas

El entrenamiento por épocas consiste en no utilizar el conjunto de entrenamiento una sola vez, si no dar varias pasadas (épocas o *epochs*), y en cada una de estas, dar pasos en el descenso de gradiente calculando este en cada subconjunto del conjunto de entrenamiento o *minibatch*

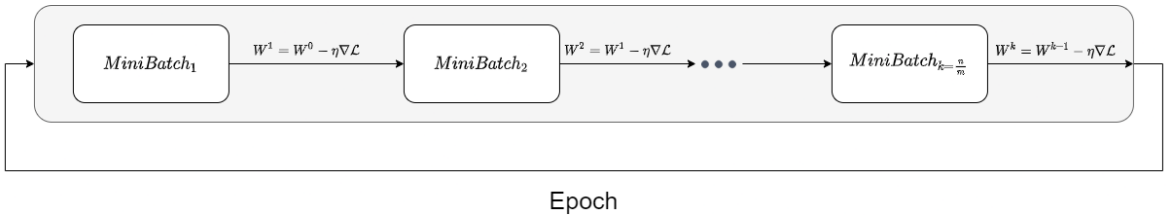

Figura 2.14: Esquema del entrenamiento por épocas y minibatches

Con esta metodología conseguimos dar varias pasadas a la muestra, por lo que el gradiente se calculará en base a pesos actualizados para toda la información contenida en el conjunto de entrenamiento, varias veces.

Callbacks

Finalmente, algunas utilidades que propociona Keras nos pueden ayudar a mejorar el entrenamiento del modelo, las Callbacks. Estas funciones se llaman en cada etapa de entrenamiento y realizan una acción de cualquier tipo. En este caso, usaremos dos de las más populares:

- **EarlyStoping**: monitorizando la curva de evolución de la pérdida en el conjunto de validación, podemos detectar cuándo ha dejado de aprender. Cuando llega a una *meseta*, se corta el entrenamiento ya que seguir entrenando incrementaría el sobreajuste.
- **ReduceLROnPlateau**: monitorizando también la evolución de la pérdida en el conjunto de validación, podemos ir reduciendo el learning rate. Al llegar a una meseta, reducir el learning rate (a la mitad en este trabajo) puede ayudar a que el descenso de gradiente nos lleve a un mínimo menor que en el que se encuentra la red en ese momento.

El factor *paciencia* determina cuándo debería considerarse que se ha llegado a una meseta. Estableciendo una paciencia de 10 (10 iteraciones sin mejorar la pérdida se considera meseta) para ReduceLROnPlateau y de 20 para EarlyStopping, damos opción a que sucedan ambos eventos. La ventaja de estas callbacks es que nos ahorra el ajustar algunos hiperparámetros como el learning rate y el número de épocas.

2.7.2. Técnicas de Regularización

De cara a reducir el sobreajuste, el Deep Learning utiliza ciertas técnicas de regularización. La idea es poner *trabas* a la red, introduciendo perturbaciones de cara a llevar la actualización de los pesos hacia los de un modelo con mayor capacidad de generalización

2.7.3. Dropout

La técnica de Dropout [33] fue introducida como un método de regularización patentado por Google. La idea es, de forma aleatoria, bloquear neuronas durante el entrenamiento en cada etapa. De este modo, se evita que la red caiga en una *obsesión* por una neurona, de cara a que se puedan encontrar patrones más complejos combinando distintas neuronas. La figura 2.15 muestra un ejemplo de cómo funciona esta técnica.

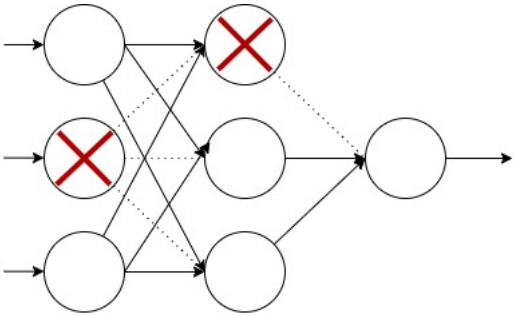

Figura 2.15: Ejemplo sobre Dropout [22]

2.7.4. Penalización L_1 y L_2

La regularización L_1 (Lasso) y L_2 (Ridge), introducen penalizaciones en el proceso de entrenamiento, en las pérdidas, de cara a reducir el sobreajuste. Se puede expresar como:

$$\mathcal{L}_{regularizada} = \mathcal{L} + \lambda_{Lasso} \sum |\theta_i| + \lambda_{Ridge} \sum \theta_i^2$$

En las redes elásticas "*clásicas*", θ son los parámetros del modelo únicamente. Sin embargo, en el Deep Learning, esta regularización se puedde aplicar siendo θ distintos parámetros:

- Kernel Regularizer: Reduce los pesos ($\theta = W$)
- Bias Regularizer: Reduce el sesgo ($\theta = b$)
- Activity Regularizer: Reduce la salida de la capa ($\theta = y = F(Wx + b)$)

2.7.5. Data Augmentation

Otra técnica muy popular en el campo de la visión computacional es el *Data Augmentation*. Consiste en la perturbación en cada época de las imágenes del conjunto entrenamiento, de cara a que la red vea en cada paso imágenes ligeramente distintas, introduciendo variedad y reduciendo el sobreajuste. Esto también aumenta la cantidad de información a la que accede la red, siendo importante en campos que no tienen acceso a los grandes volúmenes de datos que requiere el deep learning (similar a la situación en la que se enmarca este trabajo de fin de grado) [34].
Sin embargo, las técnicas más habituales de Data Augmentation están pensadas para imagenes no abstractas. Se basan en rotaciones, transformaciones espejo, aumento y

dismunución del brillo, recortes... Sin embargo, nosotros estamos trabajando con imágenes de espectrogramas, que poseen un carácter distinto y relativamente uniforme.

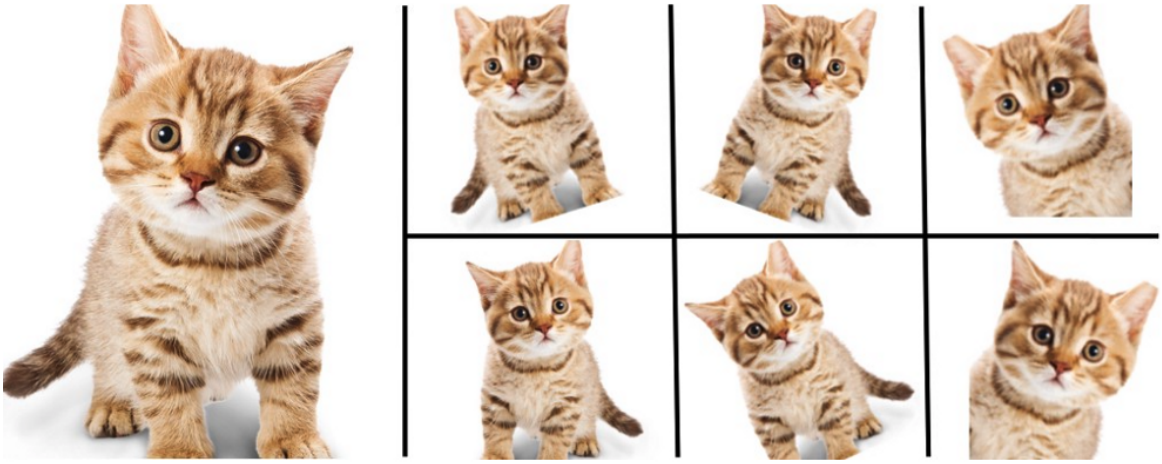

Figura 2.16: Ejemplo de *Data Augmentation*, extraído de [35]

Para este tipo de datos, existe una alternativa propuesta por Daniel S. Park y William Chan [36], en el que en vez de perturbar las señales de audio (lo que induce una perturbación en el espectrograma), perturbaron sus espectrogramas directamente, ocultando zonas del espectrograma por franjas de forma aleatoria. Siguiendo esta alternativa, en este trabajo se realizará una aproximación similar, como muestra la figura 2.17.

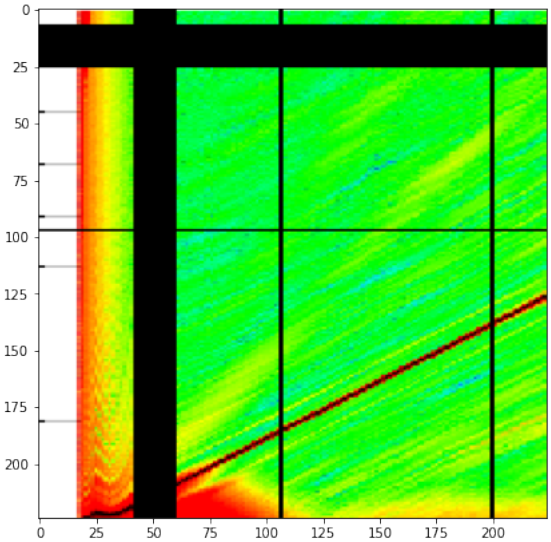

Figura 2.17: Ejemplo de *Data Augmentation*, aplicado sobre el estado transitorio de un espectrograma

2.8. Métodos de interpretabilidad

La sección se desarrolla en base a [37], centrándonos en la aplicación para imágenes que es lo que nos concierne.

2.8.1. LIME

El método *Local Interpretable Model-agnostic Explanations* o LIME [38] es uno de los métodos más populares que existen hoy en día en el ámbito de la interpretabilidad. Debido a su sencillez y rapidez, es una gran alternativa a la hora de encontrar explicaciones en la predicción de una imagen.

Según los autores, un interpretador de modelos debería tener las siguientes propiedades:

- Interpretabilidad: debería darse una relación numérica clara entre las variables predictoras y la respuesta
- Fidelidad Local: la interpretabilidad debería ser fiable, al menos de forma local, en el sentido de que para cada observación se puede obtener su explicación aunque esta no sea representativa del modelo de forma global
- *Model agnostic*: el método debería ser empleable fuese cual fuese el modelo base a interpretar
- Perspectiva global: A la hora de interpretar el modelo, la explicación debería ser precisa (en el sentido del modelo subrogado) pero a la vez ser útil para el usuario

Compromiso Fidelidad-Interpretabilidad

Sea $g \in G$ un modelo interpretable perteneciente al conjunto de modelos interpretables, el que será nuestro interpretador (véase una regresión lineal, un árbol de decisión...) para otro modelo no interpretable f. La complejidad del modelo viene definida por $\Omega(g)$, y la fiabilidad del modelo es una función $L(f, g, \pi_x)$, donde π_x es la distancia (según la métrica empleada) a la observación x a explicar. Por tanto, tenemos que buscar un punto intermedio entre L y Ω, expresando esto de tal forma que:

$$\text{interpretación}(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

Sin embargo, como el término $\Omega(g)$ es difícil de optimizar, en la práctica se fija la complejidad del modelo de antemano

Muestreo para exploración local

La idea detrás de LIME es la siguiente:

1. Tomar la observación a interpretar, cuyo formato sea interpretable (x'). En el caso de datos tabulares, no hace falta transformarla. En el caso de las imágenes como el que nos concierne, la versión interpretable de nuestra imagen x será $x' \in \{0,1\}^{d'}$, un vector binario en el que el 1 indica la presencia del píxel y 0 la no presencia

2. Perturbando esta observación, obtener un conjunto de muestras z' situada entorno a ella (en nuestro caso, ocultando y mostrando píxels de forma aleatoria), creando el conjunto $\mathcal{Z}$. Para cada z', obtenemos también $f(z')$ y $\pi_x(z')$

3. Ajustar un modelo g sobre $(\mathcal{Z}, f(\mathcal{Z}))$, con pesos $\pi_x(\mathcal{Z})$

Los autores de LIME proponen usar modelos g de tipo Lasso ya que seleccionan variables por si mismos (facilitando la interpretación) y como métrica de proximidad para los pesos un kernel exponencial:

$$\pi_x(z) = \exp\left(\frac{-D(x,z)^2}{\sigma^2}\right)$$

siendo la función objetivo del modelo g el Error Cuadrático Medio Ponderado:

$$\mathcal{L}\left(f, g, \pi_x\right) = \sum_{z,z' \in \mathcal{Z}} \pi_x(z) * \left(f(z) - g\left(z'\right)\right)^2$$

(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Figura 2.18: Ejemplo de interpretaciones LIME para tres clases sobre un clasificador de imágenes del trabajo original [38]

2.9. Grad-CAM

Otro método muy reciente de interpretabilidad popular es *Gradient-weighted Class Activation Mapping* (Grad-CAM) [39]. La idea detrás de Grad-CAM es considerar la última capa convolucional, que contiene los patrones que se alimentan a la parte densa final para la clasificación. A cada uno de los filtros (*feature maps*) de esta última capa lo llamamos $A_1, \ldots, A_F$ donde F es el número de feature maps.
Cada A_k contiene información que es relevante para todas las clases, pero nos interesa buscar qué partes de cada A_k excitaron la respuesta para una clase c concreta. Para esto, mediante el gradiente se ve para cada píxel de cada A_k en qué medida excita la clasificación como c.
Matemáticamente, la descripción es esta:

1. Propagar la imagen a explicar y obtener el valor máximo valor *crudo* (x_i en la Figura 2.6) que se pasa a la capa softmax. Llamamos a este valor crudo y^c, siendo c la clase a la que está asociado.

2. Asignar 0 a todos los valores crudos que no sean el máximo

3. Calcular el gradiente respecto a nuestros A_k, $\frac{\delta y^c}{\delta A^k}$

4. Calcular pesos para cada feature map según los pixel y su *contribución* a cada A_k para la clase c, esto es :

$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{Promedio Global}} \underbrace{\frac{\delta y^c}{\delta A_{ij}^k}}_{\text{Gradientes respecto} A_k}$$

5. Calcular la media ponderada según los α_k^c de los mapas A_k. Aplicar una capa ReLU (parte positiva) de estos mapas y escalar el resultado al intervalo [0,1]:

$$L^c_{Grad-CAM} \in \mathbb{R}^{u\times v} = \underbrace{ReLU}_{\text{Valores positivos}}\left(\sum_k \alpha_k^c A^k\right)$$

Donde $L^c_{Grad-CAM}$ será nuestro mapa de interpretación, tomando valores cercanos a 1 en las regiones que excitaron la clasificación como c.

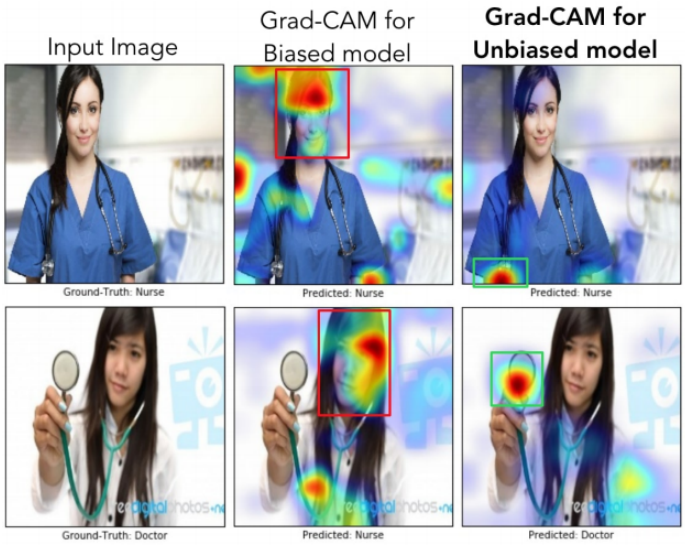

Figura 2.19: Ejemplo de cómo Grad-CAM puede ayudar a detectar sesgos, extraído de [39]

2.10. Sobre la utilidad de los métodos de interpretación

Existen otras alternativas como SHAP [40] y sus variantes, un método de interpretabilidad de los más populares basado en teoría de juegos, que se emplearon en trabajos anteriores [2] pero debido a incompatibilidades en las librerías no se han empleado en este trabajo. Es interesante destacar las dos perspectivas que aquí se toman. Por un lado tenemos métodos *Model Agnostic* como LIME y SHAP, los cuales se pueden aplicar sea cual sea el clasificador base, y por otro lado, interpretaciones *ad-hoc* como Grad-CAM, que son solo aplicables a algoritmos similares a las redes neuronales convolucionales, existiendo aún camino por recorrer en ambas alternativas.

La importancia de los métodos de interpretación no viene solo dada por el hecho de poder explicar el cómo se clasificó una observación, si no de entender si nuestro modelo está aprendiendo un concepto correcto. La imagen 2.19 muestra cómo gracias a Grad-CAM puede deducirse si un modelo está clasificando de forma correcta, o tiene algún tipo de sesgo (en el ejemplo, asocia la clase Enfermera a las regiones donde figura una mujer en el modelo sesgado, y en el modelo sin sesgo al estetoscopio). Este tipo de sesgos no siempre aparecen de forma intencionada, si no que debido a la recopilación de datos, los sesgos sociales se filtran en los conjuntos empleados para entrenar. Es por

esto que este tipo de técnicas empiezan a tener un papel relevante en la verificación de algoritmos, de cara a tener en cuenta lo que se conoce como *Algorithmic Fairness*.

3. Descripción y procesado de los datos

3.1. Descripción del conjunto de datos

Se tienen registradas 570 observaciones de la corriente de alimentación de motores de inducción sobre los cuales se realizó una perforación sobre el rotor para emular un deterioro. De cada una de estas 570 observaciones, se obtuvo el espectrograma asociado mediante la **Transformada Dragón** [6], como ilustra la Figura 2.1.

Estado	**Nº Observaciones**
R1	180
R2	93
R3	99
R4	99
R5	99

Tabla 3.1: Distribución por estado de deterioro del número de observaciones

No obstante, las imágenes fueron proporcionadas con leyenda y ejes, luego habrá que recortar estas regiones para quedarnos solo con las zonas que representan al contenido real del espectrograma, como ilustra la figura 3.1.
A pesar de ser una imagen que debería ser uniforme, en algunos espectrogramas aparece una región de color blanco. Esto representa el tiempo que tardó en arrancarse el experimento, es variable y no tiene relación con el estado de deterioro. Debido a su carácter variable, no es fácil realizar un preprocesado uniforme para todas las observaciones, luego algunos de los espectrogramas contendrán esta región blanca que se considera ruido.

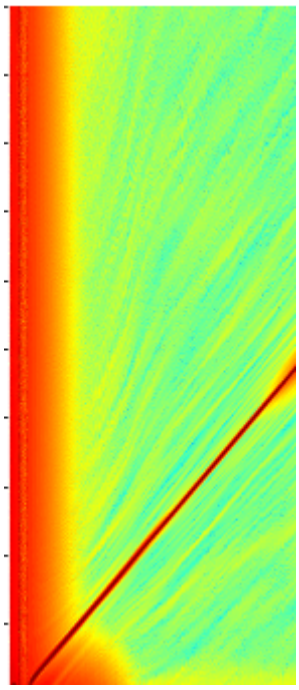

Figura 3.1: Ejemplo recorte espectrograma al estado transitorio

3.2. Muestreo de los datos y trabajo previo

Estos datos nacen de un trabajo previo del departamento de Ingeniería Eléctrica, se trata de realizar una inspección no invasiva sobre el motor estudiando la corriente de alimentación con el objetivo de conocer su estado y realizar, por tanto, un mantenimiento predictivo de los mismos. Esta técnica se conoce como *Motor Current Signature Analysis* [41]. Los datos se obtuvieron realizando perforaciones sobre el rotor del motor de distintas medidas, reflejadas en la Tabla 3.2. Además, tenemos la información del inversor que se empleó , pudiendo ser este uno de los modelos AB, ABB o TM, y el nivel de carga dos al que se realizó el experimento (NC0,NC1,NC2). El primer nivel no se usará para el estudio pormernorizado de los resultados ya que en trabajos anteriores no se empleó y representa una parte muy pequeña de la muestra.

	Perforación	
Estado	Profundidad	Diámetro
R1	0	0
R2	4.2mm	2.5mm
R3	9.4mm	2.5mm
R4	17mm	2.5mm
R5	17mm	3.5mm

Tabla 3.2: Medidas de la perforación según el estado de deterioro

La idea detrás de estas perforaciones es que cuanto mayor sea la perforación, mayores serán las intensidades de los armónicos LSH y USH inducidos sobre la frecuencia fundamental de la onda de alimentación, debido a una mayor vibración a causa de la inestabilidad estructural que estos daños producen.

La Figura 3.2 ilustra este fenómeno. Podemos ver cómo en torno a la frecuencia fundamental surgen dos frecuencias paralelas (que, en el estado transitorio, asociado a la región con una diagonal) que representan el LSH y el USH. A mayor deterioro, mayores serán los valores en estas dos franjas.

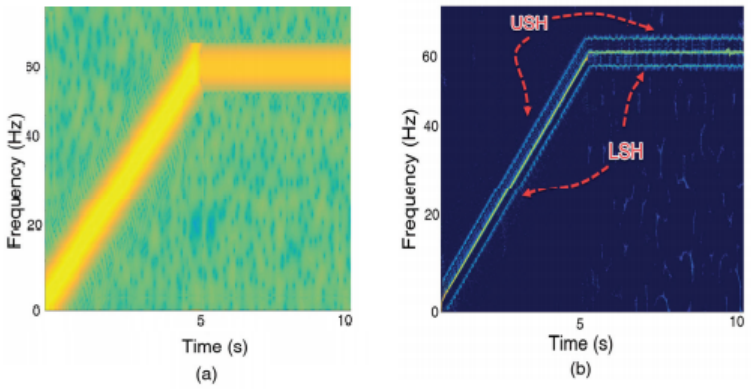

Figura 3.2: Ejemplo de los armónicos inducidos LSH y USH. En a), tenemos el espectrograma original, y en b) el espectrograma representando solo las frecuencias que superan un cierto umbral, detectándose los armónicos LSH y USH más fácilmente. Imagen de [41]

3.2.1. Trabajo previo

Se sabe de trabajos anteriores como [2, 41] que la intensidad del LSH y el USH son determinantes a la hora de predecir el estado de deterioro. Por tanto, nuestro mo-

delo debería ser capaz de encontrar patrones en estas regiones o en otros armónicos fundamentales.

Además, el objetivo de este trabajo es el de el crear un modelo que gracias a la forma en la que recibe la información, sea capaz de fijarse no solo en la intensidad de los armónicos, si no que pueda detectar relaciones espaciales entre estos gracias a que estamos utilizando información no estructurada.

4. Ajuste de Modelos

4.1. Ajuste de hiperparámetros

De cara a ajustar hiperparámetros, se realizaron pruebas usando particiones train-val-test con porcentajes (65-15-20) (sin usar la partición test) en este proceso. La toma de decisiones se ilustra en la figura 4.1, sobre los posibles valores de la tabla 4.1

Hiperparámetro	**Posibles Valores**
Nº Capas	{1,2,3,4}
Neuronas/capa	{1,2,3,4}
Dropout Rate	[0.1,0.8]
Regularización	$[10^{-5}, 10^{-3}]$
Última capa del modelo base congelada	[135-151]
Pooling a la salida del modelo base	{GlobalMaxPooling. GlovalAvgPooling}
Batch Size	{4,8,16}
Optimizer	SGD

Tabla 4.1: Posibles valores de los hiperparámetros

1) **¿Hay sobreajuste?**: En este punto se evalúa si el modelo está sobreajustando al conjunto de entrenamiento. La forma de detectar este fenómeno viene por las curvas de pérdida del modelo. En estas, dibujamos la evolución de la pérdida en cada *epoch* para el conjunto de entrenamiento y de validación, y vemos una posible divergencia entre ambas (las métricas sobre el conjunto de entrenamiento sigue mejorando pero sobre conjunto de validación no). Las Figuras 4.2 y 4.3 representan respectivamente

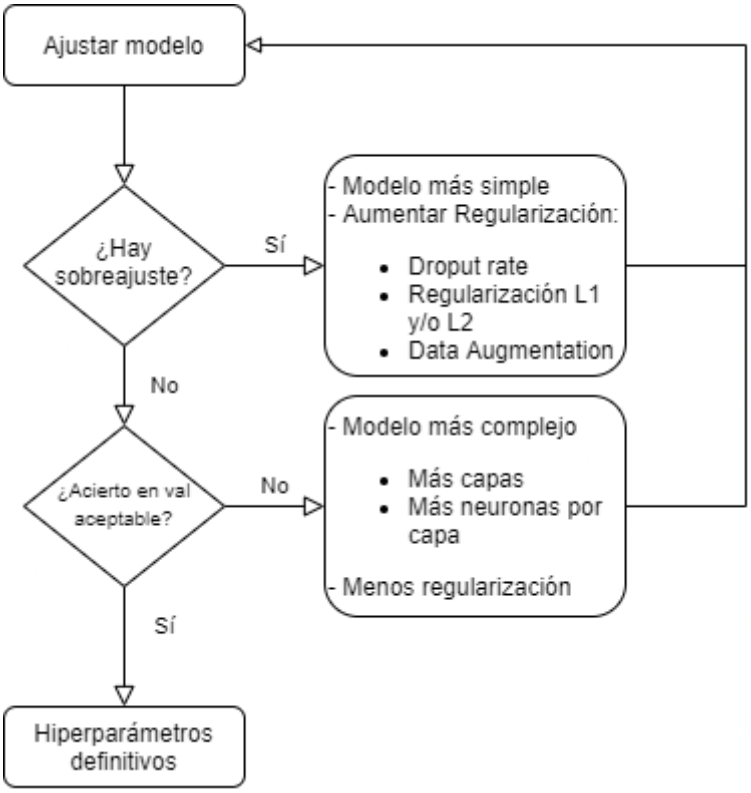

Figura 4.1: Diagrama de flujo para el ajuste de hiperparámetros

una curva de un modelo sobreajustando y de un modelo que no presenta un sobreajuste severo

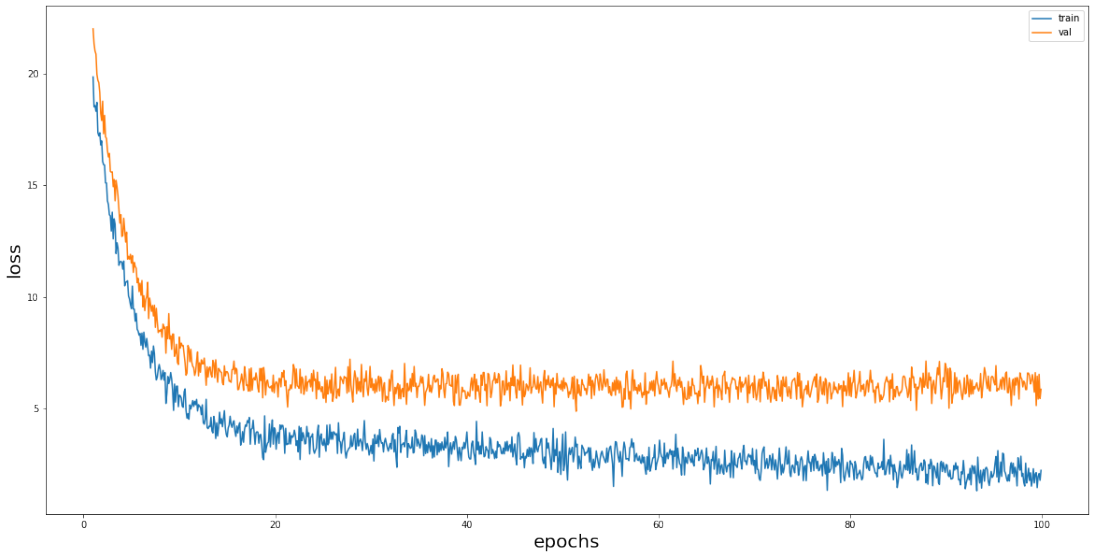

Figura 4.2: Ejemplo de curvas de aprendizaje que reflejan sobreajuste

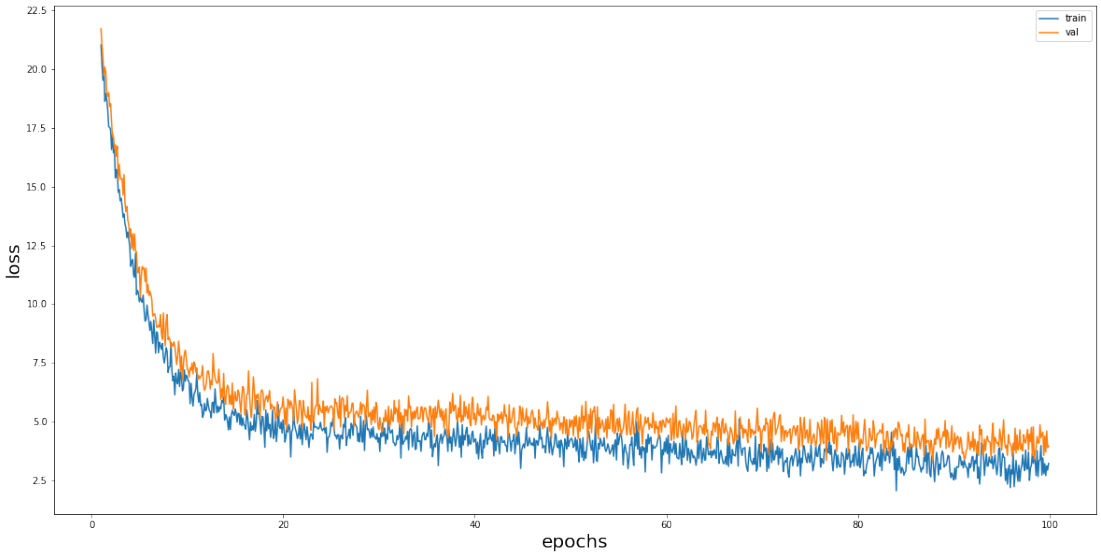

Figura 4.3: Ejemplo de curvas de aprendizaje que reflejan un buen ajuste

Para pasar de la figura 4.2 a la figura 4.3, existen distintas alternativas, como son aumentar el ratio de Dropout, la intensidad de la regularización y de la Data Augmentation.

2)**¿Acierto en validación aceptable?** Si ya se ha solucionado el sobreajuste, se decide si el modelo tiene un rendimiento aceptable, en nuestro caso en base a la tasa de acierto en la partición de validación. De trabajos anteriores, se tiene una tasa de acierto en torno al 80 %, y ese será el objetivo a alcanzar (o superar). De cara a incrementar este rendimiento, se emplea un modelo más complejo, para lo cual se puede o bien añadir más capas o más neuronas por capa o bien reducir la intensidad de las restricciones que impone la regularización sobre los pesos del modelo.

3)**Hiperparámetros definitivos**: Si ya se tiene una tasa de acierto razonable, declaramos como *ganador* al conjunto de hiperparámetros con el que hemos obtenido la mejor tasa de validación.

Cabe destacar que la búsqueda de hiperparámetros, si bien fue exhaustiva y rigurosa, se vio afectada en su profundidad por la falta de equipamiento a la hora de entrenar. El uso de GPUs acelera enormementen el entrenamiento de modelos En nuestro caso estamos usando una CPU de 16 núcleos, con lo cual queda abierto camino para mejorar el modelo probando más hiperparámetros.
Los mejores hiperparámetros fueron:

- Congelar hasta la capa 151 (dejando entrenable la última convolución)
- Añadir 2 capas con 4096 neuronas, precedidas por GlobalMaxPooling a la salida del modelo base MobileNetV2
- Dropout en cada capa con un rate de 0.3
- Regularización L2 en Kernel, Bias y Activity en todas las capas con un $\lambda = 0{,}0001$
- Batch Size = 8

Por tanto, el modelo conjunto final es el siguiente reflejado en la Tabla 4.2 (siendo las capas *bottleneck* los bloques de construcción comentados en la sección 2.5.2):

	Capa	Repeticiones	Stride	¿Entrenable?
Modelo Base	*Input*	1	-	No
	Conv2D (3x3, 32)	1	2	No
	bottleneck	2	1	No
	bottleneck	3	2	No
	bottleneck	4	2	No
	bottleneck	3	2	No
	bottleneck	3	1	No
	bottleneck	1	2	No
	Conv2D 1x1	1	1	Sí
Capas Añadidas	*GlobalMaxPooling*	1	-	Sí
	Dense (4096)	2	-	Sí
	Softmax(5)	1	-	Sí

Tabla 4.2: Topología final de la red. Repeticiones indica el número de veces que se usa la capa de forma consecutiva

4.2. Estimador final del error

Dado que estamos frente a un dataset pequeño, en aras de garantizar una buena tasa de error y no dar un único estimador puntual (podría haber sido que los hiperparámetros ajustados se hubiesen decidido en base a una partición concreta que fuese *fácil* de predecir), realizaremos el procedimiento marcado en el Algoritmo 1. De cara a paliar el problema de los mínimos locales, para cada repetición se realizan cinco arranques, quedándonos con el mejor de ellos.

Nótese que este procedimiento tiene sus pros y sus contras. Por un lado, la estimación del error es más fiable dado que estamos probando varias particiones train-val-test. Por otro lado, los hiperparámetros de la red se ajustaron para unas observaciones de una cierta partición de validación que al iterar y repetir, se verán incluídas en el test. Sin embargo, esta inclusión de observaciones de validación en el test es mínima en cada iteración y las ventajas de obtener una mejor estimación del error de generalización compensan este hecho, luego el procedimiento, a pesar de sus fallos, es mejor en nuestra opinión que el hacer una única división en train-val-test y proporcionar una estimación puntual del error.
Esto es importante de cara a evitar caer en el problema del *cherry picking* o de falacia de evidencia completa. De hecho, en los resultados del procedimiento del algoritmo 1 se obtuvieron tasas de acierto del 89 % en algún caso, pero proporcionar solo este caso como estimador del error no es honesto, a pesar de que el promedio de errores

Algorithm 1: Obtención de la tasa de error

Result: $\hat{e}$ = Estimador de la tasa de error de generalización

```
for repetición = 1..20 do

    train,val,test = Preparar_datos()
    mejor_error_val = 0.99
    mejor_modelo = NIL

    for intentos = 1..5 do

        modelo = Entrenar_Modelo(train,val)
        error_val = Error(Modelo,val)

        if error_val < mejor_error_val then
            mejor_modelo = modelo
            mejor_error_val = error_val
        end

    end

    errores(repetición) = Error(mejor_modelo,test)

end

ê=Promedio(errores)
```

proporcionará una métrica peor al incluirse *malas repeticiones*. El código necesario para este procedimiento se encuentra en los Anexos A.1 y A.2 .

4.3. Resultados

Las tablas 4.3 y 4.4 muestran las matrices de confusión (reales por filas y predichos por columnas) para las 20 iteraciones (sumadas).

Resultados Globales					
	R1	R2	R3	R4	R5
R1	639	48	15	16	2
R2	103	251	9	33	4
R3	10	16	354	29	11
R4	26	26	19	298	51
R5	3	2	6	29	380
Tasa Acierto = 0.808					

Tabla 4.3: Matriz de confusión global

Resultados Globales					
	R1	R2	R3	R4	R5
R1	0.89	0.07	0.02	0.02	0
R2	0.26	0.63	0.02	0.08	0.01
R3	0.02	0.04	0.84	0.07	0.03
R4	0.06	0.06	0.05	0.71	0.12
R5	0.01	0	0.01	0.07	0.9
Tasa Acierto = 0.808					

Tabla 4.4: Matriz de confusión global en porcentaje

Las Tablas 4.5 y 4.6 muestran los resultados obtenidos a lo largo de las repeticiones en total y en porcentaje respectivamente para cada combinación de inversor y nivel de carga.

AB-NC1					
	R1	R2	R3	R4	R5
R1	105	0	0	0	0
R2	2	30	4	14	0
R3	0	6	43	2	0
R4	0	7	0	44	0
R5	0	0	0	5	46
Tasa Acierto = 0.870					

AB-NC2					
	R1	R2	R3	R4	R5
R1	121	0	0	0	0
R2	1	53	3	7	0
R3	0	0	56	4	2
R4	0	1	5	39	17
R5	0	0	2	6	54
Tasa Acierto = 0.871					

ABB-NC1					
	R1	R2	R3	R4	R5
R1	126	5	0	1	0
R2	35	34	0	0	0
R3	4	0	64	1	0
R4	5	5	0	54	5
R5	0	0	0	0	69
Tasa Acierto = 0.85					

ABB-NC2					
NC1	R1	R2	R3	R4	R5
R1	111	7	3	5	2
R2	35	32	0	5	0
R3	0	0	73	0	0
R4	6	6	0	61	0
R5	0	0	1	1	71
Tasa Acierto = 0.83					

TM-NC1					
	R1	R2	R3	R4	R5
R1	90	13	3	6	0
R2	25	44	0	0	0
R3	0	4	54	2	0
R4	2	0	7	42	9
R5	0	0	0	3	57
Tasa Acierto = 0.79					

TM-NC2					
NC1	R1	R2	R3	R4	R5
R1	86	23	9	4	0
R2	5	58	0	4	0
R3	6	6	47	9	0
R4	9	3	4	45	7
R5	0	0	0	3	65
Tasa Acierto = 0.766					

Tabla 4.5: Matrices de confusión

AB-NC1					
	R1	R2	R3	R4	R5
R1	1	0	0	0	0
R2	0.04	0.6	0.08	0.28	0
R3	0	0.012	0.84	0.04	0
R4	0	0.14	0	0.86	0
R5	0	0	0	0.1	0.9
Tasa Acierto = 0.870					

AB-NC2					
NC1	R1	R2	R3	R4	R5
R1	1	0	0	0	0
R2	0.02	0.83	0.05	0.11	0
R3	0	0	0.09	0.06	0.03
R4	0	0.02	0.08	0.63	0.27
R5	0	0	0.03	0.1	0.87
Tasa Acierto = 0.871					

ABB-NC1					
	R1	R2	R3	R4	R5
R1	0.95	0.04	0	0.01	0
R2	0.51	0.49	0	0	0
R3	0.06	0	0.93	0.01	0
R4	0.07	0.07	0	0.78	0.07
R5	0	0	0	0	1
Tasa Acierto = 0.85					

ABB-NC2					
NC1	R1	R2	R3	R4	R5
R1	0.87	0.05	0.02	0.04	0.02
R2	0.49	0.44	0	0.07	0
R3	0	0	1	0	0
R4	0.08	0.08	0	0.84	0
R5	0	0	0.01	0.01	0.97
Tasa Acierto = 0.83					

TM-NC1					
	R1	R2	R3	R4	R5
R1	0.8	0.12	0.03	0.05	0
R2	0.36	0.64	0	0	0
R3	0	0.07	0.9	0.03	0
R4	0.03	0	0.12	0.7	0.15
R5	0	0	0	0.05	0.95
Tasa Acierto = 0.79					

TM-NC2					
	R1	R2	R3	R4	R5
R1	0.7	0.19	0.07	0.03	0
R2	0.07	0.87	0	0.06	0
R3	0.09	0.09	0.69	0.13	0
R4	0.13	0.04	0.06	0.66	0.1
R5	0	0	0	0.04	0.96
Tasa Acierto = 0.766					

Tabla 4.6: Matrices de confusión en tanto por uno

Podemos ver cómo, de forma consistente, a nivel de carga 1 se obtienen mejores tasas de acierto. Es interesante destacar que el nivel de carga es algo modificable, en el sentido de que esto indica que se puede entrenar un clasificador eficaz para nivel de carga 1 usando observaciones de nivel de carga 1.

Cabe señalar que se clasifican con una precisión elevada todos los estados de deterioro, pero parece que los que más problemas dan son el R2 (deterioro leve) y, en menor, medida el R4 (deterioro severo). Sin embargo, para el inversor AB, cada nivel de carga ayuda a separar estos estados (i.e. NC1 separa bien R4 y NC2 separa bien R2). Para el inversor TM también sucede un fenómeno parecido (a NC2 se separa bien R2 y a NC1 se separa mejor R4), mientras que para el inversor ABB no parece haber diferencias a la hora de separar el estado R2.

4.3.1. Comparación de resultados

De cara a comparar los resultados con los de [2], se seleccionará para cada inversor, el nivel de carga que mejores resultados arrojó. En aras de la uniformidad, seleccionamos el nivel de carga 1, ya que para el inversor AB es equivalente (salvo para la separación de estados R2 y R4) y para ABB y TM, proporciona las mejores tasas de acierto de forma global.

Resultados de este trabajo

AB-NC1					
	R1	R2	R3	R4	R5
R1	1	0	0	0	0
R2	0.04	0.6	0.08	0.28	0
R3	0	0.012	0.84	0.04	0
R4	0	0.14	0	0.86	0
R5	0	0	0	0.1	0.9
Tasa Acierto = 0.870					

ABB-NC1					
	R1	R2	R3	R4	R5
R1	0.95	0.04	0	0.01	0
R2	0.51	0.49	0	0	0
R3	0.06	0	0.93	0.01	0
R4	0.07	0.07	0	0.78	0.07
R5	0	0	0	0	1
Tasa Acierto = 0.85					

TM-NC1					
	R1	R2	R3	R4	R5
R1	0.8	0.12	0.03	0.05	0
R2	0.36	0.64	0	0	0
R3	0	0.07	0.9	0.03	0
R4	0.03	0	0.12	0.7	0.15
R5	0	0	0	0.05	0.95
Tasa Acierto = 0.79					

Resultados de [2]

AB-NC2					
	R1	R2	R3	R4	R5
R1	0.78	0.04	0.17	0	0.01
R2	0.09	0.76	0.15	0	0
R3	0.36	0.1	0.54	0	0
R4	0.01	0.01	0	0.98	0
R5	0	0	0	0	1
Tasa Acierto = 0.812					

ABB-NC2					
	R1	R2	R3	R4	R5
R1	0.54	0.11	0.27	0	0.08
R2	0.13	0.79	0.01	0.07	0
R3	0.16	0.01	0.83	0	0
R4	0	0	0	1	0
R5	0	0	0	0	1
Tasa Acierto = 0.832					

TM-NC2					
	R1	R2	R3	R4	R5
R1	0.94	0	0.06	0	0
R2	0.01	0.66	0.33	0	0
R3	0	0.29	0.61	0.04	0.06
R4	0	0	0	1	0
R5	0	0	0	0.02	0.98
Tasa Acierto = 0.838					

Tabla 4.7: Comparación de resultados con los obtenidos en [2]

Como se puede apreciar, los resultados mejoran en gran medida los obtenidos en trabajos anteriores para inversores AB y ABB, no así para el inversor TM. Es interesante

destacar que en la Tabla 4.5 se puede apreciar cómo para cada inversor existe al menos una combinación con el nivel de carga que permite separar mucho mejor los estados R1, R2 y R3 que cómo se separaban con las técnicas boosting anteriormente empleadas.

4.4. Métodos de Interpretabilidad

A continuación se muestran, para los métodos de interpretabilidad LIME y GradCam, los promedios de las interpretaciones píxel a píxel para los aciertos en el test de un modelo, cuyas métricas se reflejan en la Tabla 4.8 para cada estado de deterioro deterioro.

	R1	R2	R3	R4	R5
R1	32	1	1	1	1
R2	3	15	1	1	0
R3	0	2	17	1	1
R4	2	3	1	15	0
R5	0	0	1	1	19
Tasa Acierto = 0.8235					

Tabla 4.8: Matriz de confusión para el modelo usado en la interpretabilidad

4.4.1. LIME

Para las interpretaciones LIME (siendo estas las regiones que favorecieron la clasificación con el estado que corresponde, existe la posibilidad de mostrar las que excitaban la no-clasificación pero nos centraremos solo en las positivas) de la figura 4.4, podemos ver cómo cada estado de deterioro el modelo puso hincapié en una región distinta:

- Para los estados **R1** y **R2**, el énfasis se sitúa en las regiones superior-derecha y superior-central respectivamente, evolucionando hacia el centro a medida que avanza el deterioro, lo que correspondería a armónicos de frecuencia alta en los momentos finales y centrales del transitorio. Cabe destacar que para el estado R1 el modelo parece estar fijándose en un entorno del armónico principal en los momentos iniciales del transitorio con bastante intensidad (región roja abajo a la izquierda), no tanto para el estado R2.

- Para el estado **R3**, el foco parece estar en gran medida en la zona final del transitorio, en los armónicos situados entorno a la región correspondiente a los *Lower* y *Upper Sideband Harmonics* empleados en trabajos anteriores.

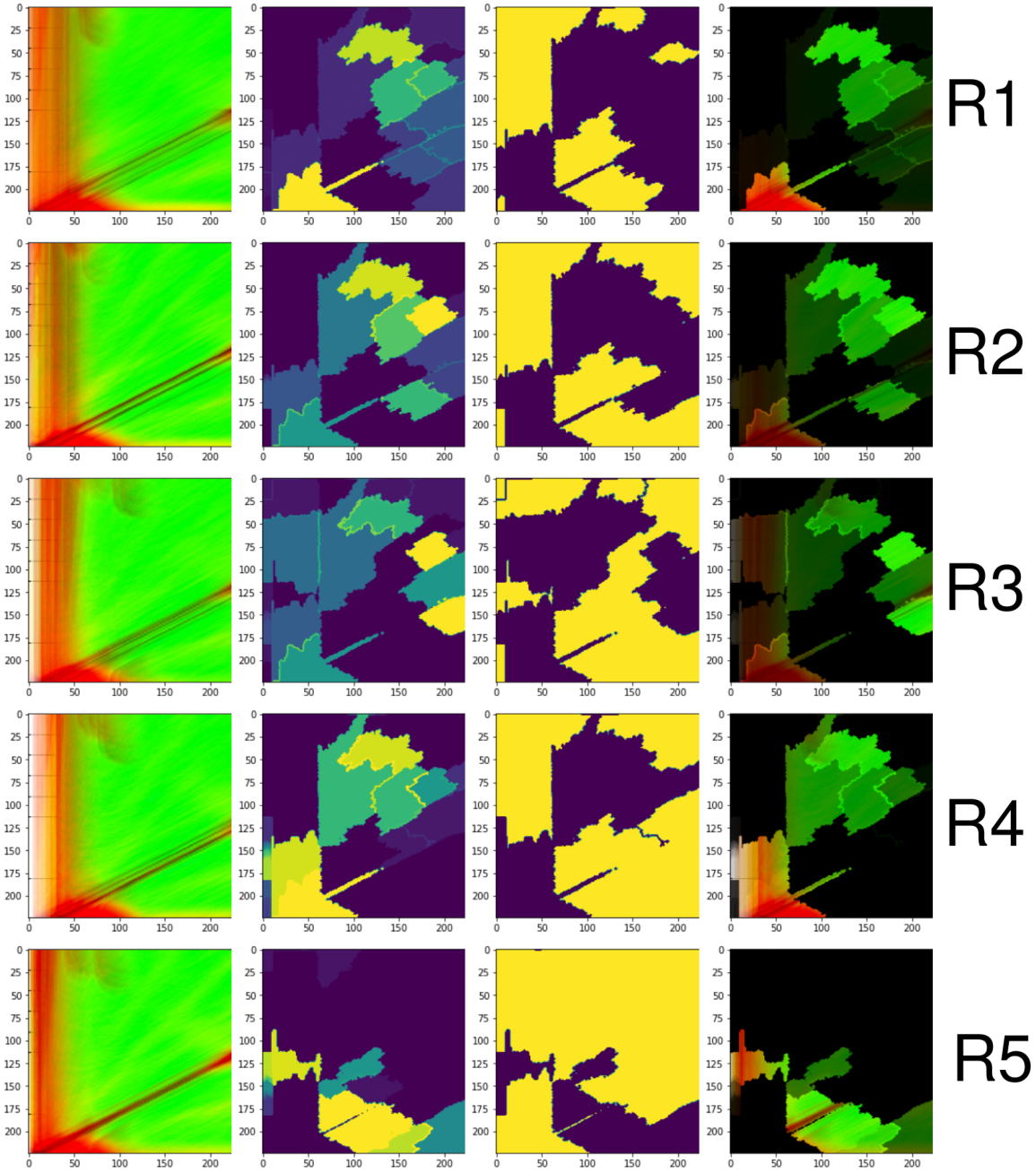

Figura 4.4: Promedios de las interpretaciones LIME sobre los aciertos del conjunto test. La primera columna muestra el promedio de la región del espectrograma asociada al estado transitorio. La segunda, los parches de píxeles según su importancia (a mayor intensidad hacia el amarillo, más veces fue relevante ese píxel. La tercera, en color morado indica si el píxel fue relevante para la clasificación en alguna imagen y en amarillo si no). La cuarta representa una combinación de la primera y la segunda columna

- Para el estado **R4**, parece estar fijándose en regiones que se solapan con las del estado **R2**, aunque parece estar poniendo énfasis en las regiones iniciales del transitorio para armónicos de frecuencia baja.
- Para el estado **R5**, se aprecia de forma patente que el modelo está centrándose en una región *exclusiva*, la correspondiente a los armónicos de frecuencia baja en los estados centrales y finales del transitorio.

4.4.2. LIME por inversor

Las Figuras 4.5 y 4.6 muestran las interpretaciones LIME de los aciertos en el test para cada combinación de inversor y estado de deterioro. El objetivo de esta sección es comprobar si el modelo se fija en distintas regiones para cada tipo de inversor. Como se puede apreciar, las diferencias no son muy relevantes, quizás en el inversor AB se aprecia que para el estadoo R3 tiene más peso la región central y para el inversor TM en el estado R4, la región superior derecha. No obstante, debido a la divisón en los inversores, el número de espectrogramas para cada combinación no es alto para ninguna combinación, luego si nos fijamos solo en las regiones relevantes, se ve que no hay diferencias según el inversor.

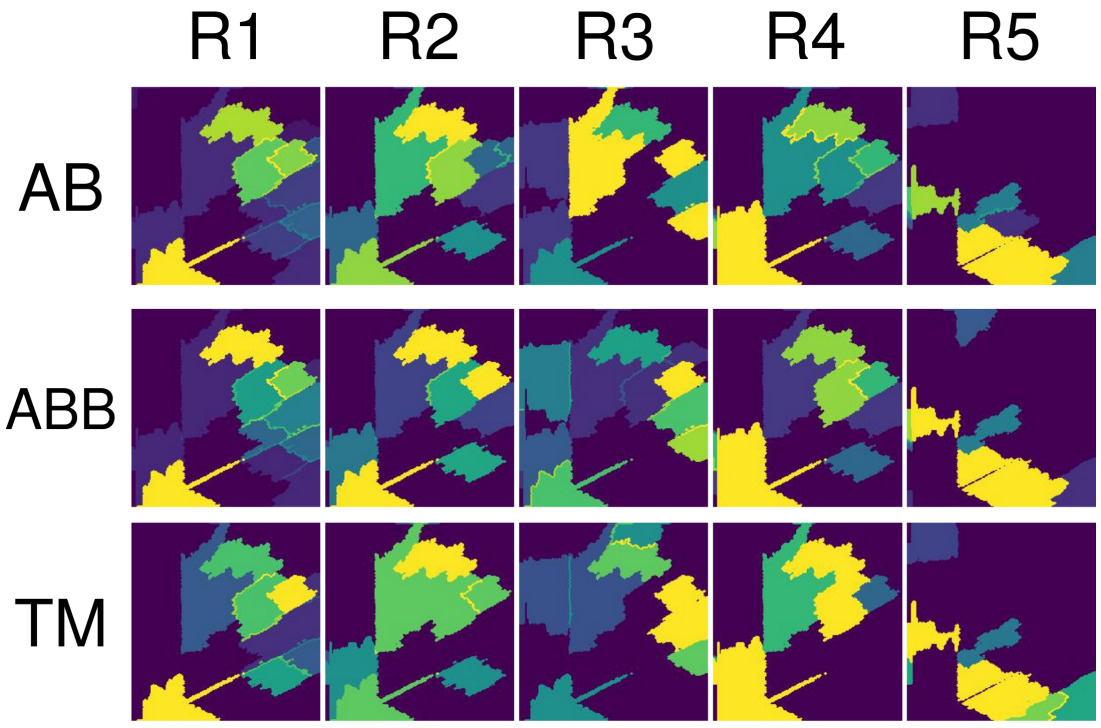

Figura 4.5: Interpretaciones LIME separadas para cada inversor

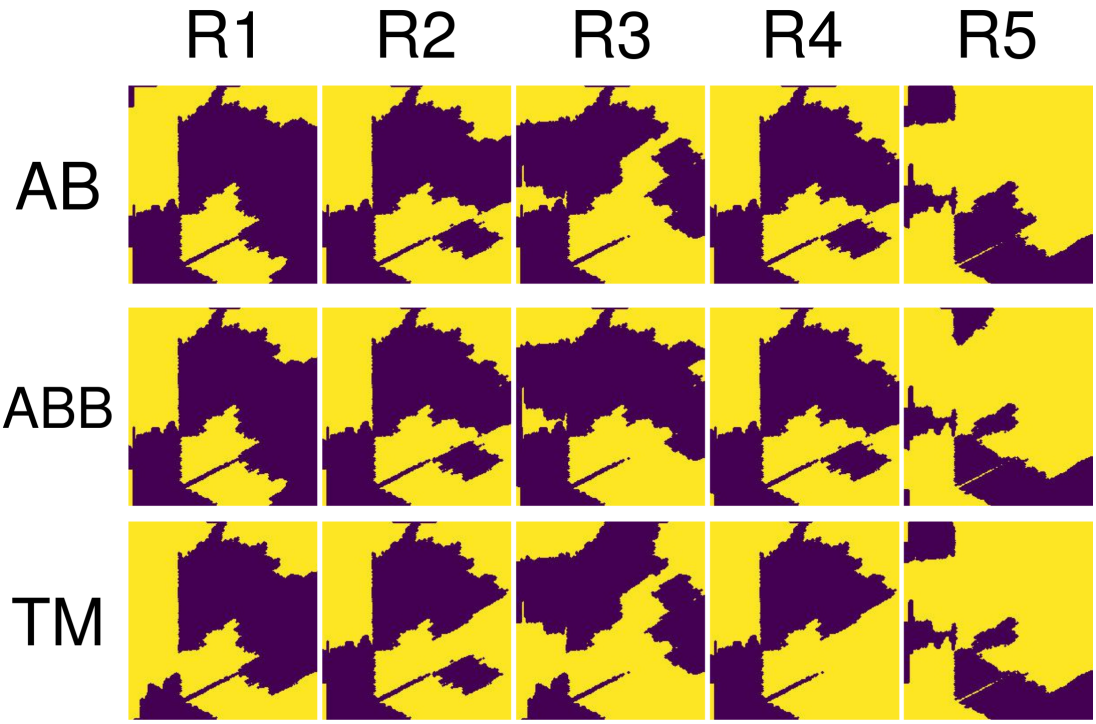

Figura 4.6: Interpretaciones LIME separadas para cada inversor (II)

4.4.3. Grad-CAM

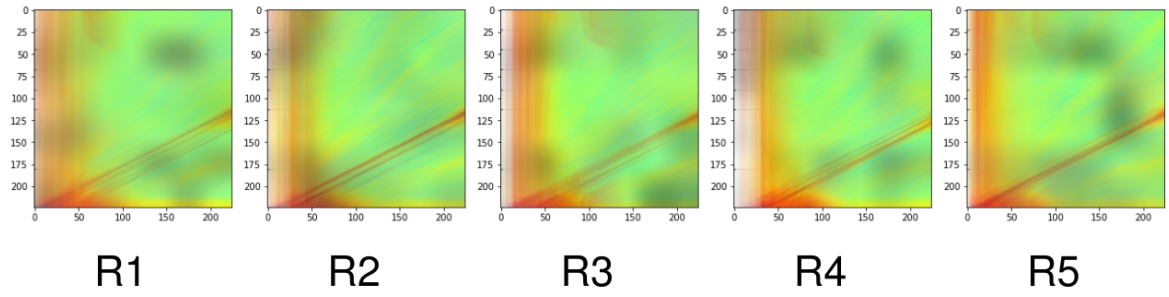

Figura 4.7: Promedios de las interpretaciones Grad-CAM sobre los aciertos del conjunto test

Lo que podemos ver, sombreado en negro, son las regiones que excitaron la respuesta. Si nos fijamos por estados de deterioro, podemos encontrar similitudes con los resultados que ofrece LIME:

1. **R1**: podemos ver como regiones en el estado central del transitorio para frecuencias altas (en el centro arriba de la imagen excitaron la respuesta)

2. **R2**: en el estado R2, no parece haber una región clara asociada según Grad-CAM. La zona con mayor influencia parece ser el inicio del estado transitorio, incluyendo la región asociada al efecto borde. Esto puede ser síntoma de que las tasas de acierto sean más bajas para este estado de deterioro.

3. **R3**: En el estado R3, parece estar señalando a los estados finales del transitorio y regiones situadas en torno al armónico principal.

4. **R4**: En el estado R4, parece estar señalando regiones similares a las de R3 pero con una localización ligeramente distinta. Mientras que para el estado R3 localiaba como influyente la región inferior izquierda, para el estado R4 detecta copmo influyente justo la región que no es influyente para R3, esto es interesante de cara a estudiar qué fenómenos incurren en esa zona en términos de frecuencia. Además, detecta como influyentes otras dos zonas en la parte superior, que vuelven a ser las que no señala R3. Es interesante destacar que las regiones son similares para R1 y R4, pero según las matrices de conffusión no parece haber una gran confusión entre estas dos clases

5. **R5**: Finalmente, para R5 Grad-CAM apunta a una región que no se señala para ningún otro estado y de forma bastante clara, situada casi al final del transitorio y en frecuencias medias, y en los estados intermedios del transitorio para frecuencias altas

El código necesario para la interpretación de modelos se encuentra en los anexos A.3 y A.4.

4.5. Resultados añadiendo el estado estacionario

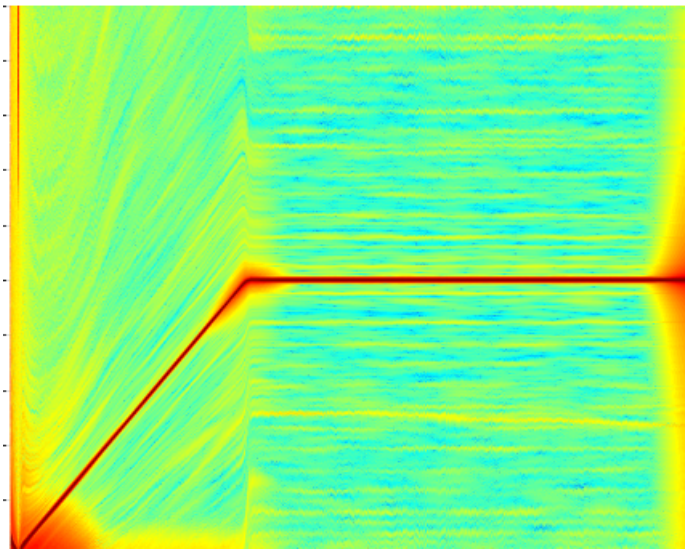

Figura 4.8: Ejemplo recorte espectrograma al estado transitorio+estacionario

Por último, se empleó el mismo procedimiento con los mismos hiperparámetros añadiendo al tensor de entrada la parte correspondiente al estado estacionario del motor, como ilustra la Figura 4.8, ya que los resultados deberían ser mejores por el carácter estable de esta etapa del arranque. Los resultados se reflejan en la Tabla 4.9

AB-NC1					
	R1	R2	R3	R4	R5
R1	0.99	0.01	0	0	0
R2	0.04	0.54	0.23	0.19	0
R3	0	0.19	0.79	0.01	0
R4	0	0.01	0	0.9	0
R5	0	0	0	0.12	0.88
Tasa Acierto = 0.847					

AB-NC2					
	R1	R2	R3	R4	R5
R1	1	0	0	0	0
R2	0.08	0.7	0.08	0.13	0
R3	0.2	0.07	0.83	0.07	0
R4	0.02	0	0.09	0.89	0
R5	0	0	0	0.06	0.94
Tasa Acierto = 0.894					

ABB-NC1					
	R1	R2	R3	R4	R5
R1	0.77	0.13	0.04	0.06	0
R2	0.43	0.56	0.02	0	0
R3	0.05	0.05	0.89	0	0
R4	0.05	0	0	0.95	0
R5	0	0	0	0	1
Tasa Acierto = 0.824					

ABB-NC2					
	R1	R2	R3	R4	R5
R1	0.77	0.13	0.04	0.06	0
R2	0.51	0.43	0.01	0.04	0
R3	0	0	1	0	0
R4	0.04	0	0.01	0.94	0
R5	0	0	0	0.03	0.97
Tasa Acierto = 0.818					

TM-NC1					
	R1	R2	R3	R4	R5
R1	0.92	0.05	0.02	0.02	0
R2	0.31	0.68	0	0.01	0
R3	0.12	0.08	0.73	0.06	0
R4	0.03	0	0.11	0.83	0.03
R5	0	0	0	0	1
Tasa Acierto = 0.843					

TM-NC2					
	R1	R2	R3	R4	R5
R1	0.72	0.09	0.02	0	0
R2	0.28	0.54	0.13	0.04	0
R3	0.47	0.04	0.44	0.04	0
R4	0.16	0.06	0.11	0.67	0
R5	0	0	0	0.1	0.9
Tasa Acierto = 0.662					

Tabla 4.9: Matrices de confusión en tanto por uno añadiendo el estado estacionario

Se puede apreciar una mejoría en los resultados para los inversores AB, donde se alcanza una tasa de error de 0.894 y en el inversor TM, donde se alcanza una tasa de acierto de 0.843. Cabe señalar que en la mayoría de aplicaciones industriales se emplea esta etapa ya que, como se decía antes, sus características estables facilitan la predicción, y estos resultados apuntan en esa dirección. Además, los hiperparámetros empleados eran los óptimos cuando solo usábamos el transitorio, luego una búsqueda de hiperparámetros personalizada debería mejorar aún más los resultados obtenidos con la parte estacionaria. Estos resultados son prometedores y apuntan a un gran abanico de opciones de cara a futuro en el mantenimiento predictivo, ya que apuntan en la dirección de que el estado transitorio pueda contener información valiosa para esta tarea.

4.6. Resultados utilizando solo el estado estacionario

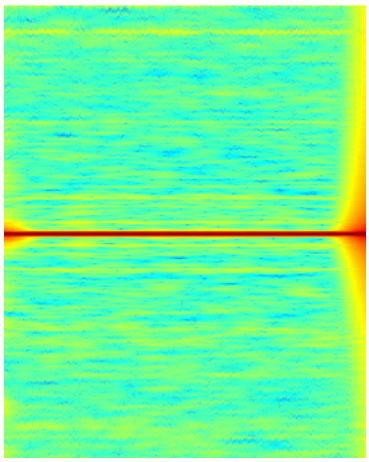

Figura 4.9: Ejemplo recorte espectrograma al estado estacionario

En este caso, los resultados (Tabla 4.10) son mucho peores. Esto no quiere decir que el estado estacionario no contenga información relevante o que sea más difícil predecir, si no que los hiperparámetros y arquitectura del modelo que hemos elegido era óptima para el estado transitorio, lo que resalta las diferencias entre los dos estados. Hay que notar también que el hecho de que las arquitecturas de las redes tengan un peor rendimiento en otro caso distinto al que se ajustaron, justifica el trabajo desarrollado en este TFM de definir un método específico para el estado transitorio.

AB-NC1					
	R1	R2	R3	R4	R5
R1	0.96	0.04	0	0	0
R2	0.07	0.39	0.31	0.23	0
R3	0.05	0.38	0.55	0.02	0
R4	0	0.07	0.02	0.91	0
R5	0	0	0.02	0.07	0.91
Tasa Acierto = 0.778					

AB-NC2					
NC1	R1	R2	R3	R4	R5
R1	0.98	0	0.02	0	0
R2	0.1	0.69	0.21	0	0
R3	0.22	0.18	0.52	0.08	0
R4	0	0	0.08	0.92	0
R5	0	0	0	0.02	0.98
Tasa Acierto = 0.851					

ABB-NC1					
	R1	R2	R3	R4	R5
R1	0.75	0.13	0.12	0	0
R2	0.5	0.42	0.01	0.07	0
R3	0.08	0.01	0.75	0.07	0
R4	0.17	0.19	0	0.64	0
R5	0	0	0	0	1
Tasa Acierto = 0.776					

ABB-NC2					
NC1	R1	R2	R3	R4	R5
R1	0.86	0.09	0.02	0.03	0
R2	0.43	0.41	0.08	0.08	0
R3	0.11	0.04	0.81	0.04	0
R4	0.28	0.02	0	0.7	0
R5	0	0	0	0	1
Tasa Acierto = 0.731					

TM-NC1					
	R1	R2	R3	R4	R5
R1	0.83	0.09	0.07	0.01	0
R2	0.66	0.34	0	0	0
R3	0.17	0.02	0.64	0.16	0.02
R4	0	0.02	0.14	0.79	0.05
R5	0	0	0	0	1
Tasa Acierto = 0.731					

TM-NC2					
NC1	R1	R2	R3	R4	R5
R1	0.76	0.14	0.01	0	0
R2	0.47	0.31	0.19	0.03	0
R3	0.61	0.12	0.25	0.02	0
R4	0.15	0.12	0.22	0.44	0.07
R5	0	0	0	0	1
Tasa Acierto = 0.585					

Tabla 4.10: Matrices de confusión en tanto por uno empleando solo la información del estacionario

5. Conclusiones y trabajo futuro

A continuación se exponen las conclusiones obtenidas a partir del estudio de este TFM.

5.1. Conclusiones

- Se ha demostrado la viabilidad y mayor eficiencia en nuestros experimentos de esta nueva metodología para mantenimiento predictivo basada en visión por computador, frente a la aproximación de Machine Learning clásico que se sigue por norma en el campo.
- El uso de información no estructurada ayuda a capturar patrones que se pierden al pasar a información estructurada o tabular. Además, el preprocesado (en este caso, seleccionar los armónicos LSH y USH) no es necesario, ni técnicas de redución de la dimensionalidad como las empleadas en [2].
- Los resultados obtenidos en este trabajo superan con creces los obtenidos en trabajos anteriores, llegando a obtener tasas de acierto globales del 85 % en algunas de las repeticiones, y del 87 % para combinaciones concretas de inversor y nivel de carga sobre cinco estados de deterioro. Además, estos resultados se obtuvieron empleando únicamente el estado transitorio, lo que a priori es una tarea más complicada ya que la onda sobre la que se calcula el espectrograma no es estacionaria.
- Se han obtenido mejores resultados a NC1 que a NC2. Según los expertos del Departamento de Ingeniería Eléctrica, es más fácil detectar deterioros a niveles de carga más altos, con [6] indicando lo mismo. Además, según el nivel de carga, puede haber una separación más sencilla para los estados R2 o R4.

- Se ha comprobado empíricamente cómo la adición de información del estado estacionario mejora los resultados aún cuando el procedimiento se estableció de forma que fuese óptimo para el uso en exclusiva del estado transitorio (esto es, a pesar de seguir utilizando los mismos hiperparámetros óptimos solo para el transitorio).

5.2. Trabajo futuro

- Estudio y empleo de nuevas técnicas y arquitecturas de Deep Learning, como el uso de topologías de tipo Autoencoder o el uso de nuevas arquitecturas como Transformers.
- Empleo de otros modelos base. A pesar de que en las pruebas iniciales se emplearon modelos como Xception o modelos tipo ResNet, se desecharon por tiempos de cómputo y por tasas de error más bajas de primeras que las de los modelos MobileNetV2. No obstante, con ordenadores más potentes, se podrían estudiar estos modelos más a fondo.
- Uso de *deep features* con otro clasificador. Una aplicación interesante subproducto del transfer learning es, una vez tenemos una red neuronal entrenada, podemos utilizar una de las capas intermedias como entrada a un clasificador de Machine Learning clásico, como un SVM o un XGBoost. Estas capas intermedias se pueden entender como una representación abstracta de la observación de entrada, y combinando esto con otro clasificador se pueden obtener buenos resultados. Especialmente interesante es el uso de esta técnica combinada con Autoencoders o redes Siamesas.
- Recolección de más datos, ya que la muestra empleada era pequeña (500 imágenes) para lo que suele ser empleado en un proyecto de Deep Learning.
- Búsqueda más exhaustiva de hiperparámetros, utilizando máquinas más potentes para facilitar la tarea.
- Uso de redes híbridas, empleando por un camino la solución aquí ilustrada basada en imágenes y por otro el paso de información del tipo inversor y nivel de carga y combinando ambas entradas.
- Depuración de los datos de entrada. Dado que los datos proporcionados fueron imágenes y no arrays numéricos, usando los segundos se puede resolver de forma sencilla el problema de las regiones blancas al comienzo del espectrograma.

- Empleo de metodología alternativa sobre el espectrograma como [42] en el que extraen métricas a partir del espectrograma y las emplean como entrada a un clasificador de tipo SVM (un paso de información no estructurada a estructurada).

Bibliografía

[1] José Ignacio Fernández Villafáñez. "Diagnóstico de fallos en rodamientos de motores eléctricos mediante técnicas lasso". *Trabajos de Fin de Grado UVa* (2018). URL: http://uvadoc.uva.es/handle/10324/30854.

[2] Alejandro Barón García. "Detección y clasificación de fallos en motores mediante procedimientos Boosting". *Trabajos de Fin de Grado UVa* (2020). URL: http://uvadoc.uva.es/handle/10324/43777.

[3] Sercan O. Arik y Tomas Pfister. *TabNet: Attentive Interpretable Tabular Learning*. 2020. arXiv: 1908.07442 [cs.LG].

[4] Vanessa Fernandez-Cavero y col. "A Comparison of Techniques for Fault Detection in Inverter-Fed Induction Motors in Transient Regime". *IEEE Access* 5 (2017). ISSN: 21693536. DOI: 10.1109/ACCESS.2017.2702643.

[5] Tensorflow-Google. *Yamnet*. URL: https://tfhub.dev/google/yamnet/1 (visitado 30-04-2021).

[6] Vanesa Fernandez-Cavero y col. "Diagnosis of Broken Rotor Bars during the Startup of Inverter-Fed Induction Motors Using the Dragon Transform and Functional ANOVA". *Applied Sciences* 11.9 (2021). ISSN: 2076-3417. DOI: 10.3390/app11093769. URL: https://www.mdpi.com/2076-3417/11/9/3769.

[7] Gopal Chandra Jana, Ratna Sharma y Anupam Agrawal. "A 1D-CNN-Spectrogram Based Approach for Seizure Detection from EEG Signal". *Procedia Computer Science*. Vol. 167. 2020. DOI: 10.1016/j.procs.2020.03.248.

[8] Helin Wang, Yuexian Zou y Dading Chong. *Acoustic scene classification with spectrogram processing strategies*. 2020.

[9] Ahmad Azab y Mahmoud Khasawneh. "MSIC: Malware Spectrogram Image Classification". *IEEE Access* 8 (2020). ISSN: 21693536. DOI: 10.1109/ACCESS.2020.2999320.

[10] Angel Sapena-Bano y col. "Harmonic Order Tracking Analysis: A Speed-Sensorless Method for Condition Monitoring of Wound Rotor Induction Generators". *IEEE Transactions on Industry Applications* 52.6 (2016). ISSN: 00939994. DOI: 10.1109/TIA.2016.2597134.

[11] Huitaek Yun y col. "Development of internal sound sensor using stethoscope and its applications for machine monitoring". *Procedia Manufacturing*. Vol. 48. 2020. DOI: 10.1016/j.promfg.2020.05.147.

[12] Moslem Azamfar y col. "Multisensor data fusion for gearbox fault diagnosis using 2-D convolutional neural network and motor current signature analysis". *Mechanical Systems and Signal Processing* 144 (2020). ISSN: 10961216. DOI: 10.1016/j.ymssp.2020.106861.

[13] Martin Valtierra-Rodriguez y col. "Convolutional neural network and motor current signature analysis during the transient state for detection of broken rotor bars in induction motors". *Sensors (Switzerland)* 20.13 (2020). ISSN: 14248220. DOI: 10.3390/s20133721.

[14] P. Arun, S. Abraham Lincon y N. Prabhakaran. "An automated method for the analysis of bearing vibration based on spectrogram pattern matching". *Journal of Applied Research and Technology* 17.2 (2019). ISSN: 16656423. DOI: 10.22201/icat.16656423.2019.17.2.805.

[15] Wo Jae Lee y col. "Learning via acceleration spectrograms of a DC motor system with application to condition monitoring". *International Journal of Advanced Manufacturing Technology* 106.3-4 (2020). ISSN: 14333015. DOI: 10.1007/s00170-019-04563-8.

[16] Seonwoo Lee y col. "A study on deep learning application of vibration data and visualization of defects for predictive maintenance of gravity acceleration equipment". *Applied Sciences (Switzerland)* 11.4 (2021). ISSN: 20763417. DOI: 10.3390/app11041564.

[17] Arka Bera, Arindam Dutta y Ashis K. Dhara. "Deep learning based fault classification algorithm for roller bearings using time-frequency localized features". *Proceedings - IEEE 2021 International Conference on Computing, Communication, and Intelligent Systems, ICCCIS 2021*. 2021. DOI: 10.1109/ICCCIS51004.2021.9397072.

[18] Warren S. McCulloch y Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". *The Bulletin of Mathematical Biophysics* 5.4 (dic. de 1943), págs. 115-133. DOI: 10.1007/bf02478259. URL: https://doi.org/10.1007/bf02478259.

[19] A G Ivakhnenko y V G Lapa. *Cybernetics and forecasting techniques*. Mod. Analytic Comput. Methods Sci. Math. Trans. from the Russian, Kiev, Naukova Dumka, 1965. New York, NY: North-Holland, 1967. URL: https://cds.cern.ch/record/209675.

[20] David E. Rumelhart, Geoffrey E. Hinton y Ronald J. Williams. "Learning representations by back-propagating errors". *Nature* 323.6088 (oct. de 1986), págs. 533-536. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[21] Alex Krizhevsky, Ilya Sutskever y Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., 2012, págs. 1097-1105.

[22] Alejandro Barón García. "Energy load forecast in smart buildings with deep learning techniques". *Trabajos de Fin de Grado UVa* (2020). URL: http://uvadoc.uva.es/handle/10324/44114.

[23] Richard H. R. Hahnloser y col. "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". *Nature* 405.6789 (jun. de 2000), págs. 947-951. DOI: 10.1038/35016072. URL: https://doi.org/10.1038/35016072.

[24] George E. Dahl, Tara N. Sainath y Geoffrey E. Hinton. "Improving deep neural networks for LVCSR using rectified linear units and dropout". *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (2013), págs. 8609-8613.

[25] Timm Dettmers. *Deep Learning in a Nutshell: Core Concepts*. Ago. de 2020. URL: https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts.

[26] Santisudha Panigrahi y Tripti Nanda AnujaSwarnkar. "A Survey on Transfer Learning". Vol. 194. 2021. DOI: 10.1007/978-981-15-5971-6_83.

[27] Keras. *Keras Applications*. URL: https://keras.io/api/applications/.

[28] Jia Deng y col. "Imagenet: A large-scale hierarchical image database". *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, págs. 248-255.

[29] Mark Sandler y col. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[30] Krzysztof Wolk y Agnieszka Wolk. "Early and Remote Detection of Possible Heartbeat Problems With Convolutional Neural Networks and Multipart Interactive Training". *IEEE Access* 7 (2019), págs. 145921-145927. DOI: 10.1109/access.2019.2919485. URL: https://doi.org/10.1109/access.2019.2919485.

[31] Ashia C. Wilson y col. *The Marginal Value of Adaptive Gradient Methods in Machine Learning*. 2018. arXiv: 1705.08292 [stat.ML].

[32] Pan Zhou y col. *Towards Theoretically Understanding Why SGD Generalizes Better Than ADAM in Deep Learning*. 2020. arXiv: 2010.05627 [cs.LG].

[33] Nitish Srivastava y col. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *J. Mach. Learn. Res.* 15.1 (ene. de 2014), págs. 1929-1958. ISSN: 1532-4435.

[34] Connor Shorten y Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". *Journal of Big Data* 6.1 (jul. de 2019). DOI: 10.1186/s40537-019-0197-0. URL: https://doi.org/10.1186/s40537-019-0197-0.

[35] Raj Bharath. *Data Augmentation: How to use Deep Learning when you have Limited Data*. URL: https://www.kdnuggets.com/2018/05/data-augmentation-deep-learning-limited-data.html.

[36] Daniel S. Park y col. "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition". *Interspeech 2019* (sep. de 2019). DOI: 10.21437/interspeech.2019-2680. URL: http://dx.doi.org/10.21437/Interspeech.2019-2680.

[37] Christoph Molnar. *Interpretable Machine Learning. A Guide for Making Black Box Models Explainable*. https://christophm.github.io/interpretable-ml-book/. 2019.

[38] Marco Tulio Ribeiro, Sameer Singh y Carlos Guestrin. ""Why should i trust you?."Explaining the predictions of any classifier". *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016. ISBN: 9781450342322. DOI: 10.1145/2939672.2939778.

[39] Ramprasaath R. Selvaraju y col. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization". *International Journal of Computer Vision* 128.2 (oct. de 2019), págs. 336-359. ISSN: 1573-1405. DOI: 10.1007/s11263-019-01228-7. URL: http://dx.doi.org/10.1007/s11263-019-01228-7.

[40] Scott Lundberg y Su-In Lee. *A Unified Approach to Interpreting Model Predictions*. 2017. arXiv: 1705.07874 [cs.AI].

[41] A. Garcia-Perez y col. "Broken rotor bar detection in inverter-fed induction motors by time-corrected instantaneous frequency spectrogram". *2017 IEEE 11th International Symposium on Diagnostics for Electrical Machines, Power Electronics and Drives (SDEMPED)*. 2017, págs. 280-285. DOI: 10.1109/DEMPED.2017.8062368.

[42] Qinyu Jiang, Faliang Chang y Chunsheng Liu. "A Spectrogram Based Local Fluctuation Feature for Fault Diagnosis with Application to Rotating Machines". *Journal of Electrical Engineering and Technology* (2021). ISSN: 20937423. DOI: 10.1007/s42835-021-00704-w.

A. Anexos

A.1. train.py

```
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications.mobilenet import preprocess_input
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D,
    Dropout,Activation,BatchNormalization,LeakyReLU,GlobalMaxPooling2D

from tensorflow.keras.optimizers import SGD,Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau,EarlyStopping

from tensorflow.keras.regularizers import l1,l2,l1_l2
from tensorflow.keras.optimizers import SGD

from utils import prepare_data
from utils import split_folders
from utils import get_generators

from sklearn.metrics import confusion_matrix

import splitfolders
import numpy as np
import pickle as pkl
import itertools
import os
```

```
28 import sys
29
30 rep_name = sys.argv[1]
31 INVS = ['AB','ABB','TM1']
32 NCS = ["NC1","NC2"]
33
34 COMBINATIONS = set([l for x in itertools.permutations(INVS,len(NCS))
       for l in list(zip(x,NCS))])
35
36
37 '''
38 Model definition function:
39
40 - Frozen MobilenetV2 as base model
41 - Adds some layers on top and a softmax nclass layer at the end
42 '''
43 def myModel(nclass=5,freeze_base=True):
44     base_model = MobileNetV2(weights='imagenet', include_top=False)
45     b = 0.0001
46     reg = l1_l2(b,b)
47     drop_rate = 0.3
48
49     regs = {"kernel_regularizer":reg,
50             "bias_regularizer":reg,
51             "activity_regularizer":reg}
52     x = base_model.output
53
54
55     x = GlobalMaxPooling2D()(x)
56     x = Dropout(drop_rate)(x)
57
58     x = Dense(4096,**regs)(x)
59     x = LeakyReLU(0.03)(x)
60     x = Dropout(drop_rate)(x)
61
62     x = Dense(4096,**regs)(x)
63     x = LeakyReLU(0.03)(x)
64     x = Dropout(drop_rate)(x)
65
66     predictions = Dense(nclass, activation='softmax')(x)
67
68     model = Model(inputs=base_model.input, outputs=predictions)
69
70     if freeze_base:
```

```
        for layer in base_model.layers:
            layer.trainable = False

    return model

'''
Performs a training stage (meaning a full epochs cicle)

- Batch size is ignored when using generators

'''
def train_stage(model,train_generator,validation_generator,batch_size
    =8,epochs=10,lr=0.0005,verbose=0):

    reduce_lr = ReduceLROnPlateau(monitor ='val_loss',
                                  factor = 0.5,
                                  patience = 10,verbose=2)

    early_stop = EarlyStopping(monitor = "val_loss",
                               patience =20,
                               restore_best_weights=True)

    optim = SGD(lr=lr, momentum=0.9)

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

    class_weight = {0: 0.75,
                    1: 1.,
                    2: 1.,
                    3: 1.,
                    4: 1.,}

    model.fit(train_generator,
              batch_size=batch_size,
              epochs=epochs,
              validation_data=validation_generator,
              callbacks = [reduce_lr,early_stop],
              class_weight=class_weight,
              verbose = verbose)

'''
Retrieves, given a generator, a confusion matrix for each Inversor and
    Charge Level based on the filenames
```

```
'''
def subset_matrix(inversor,nc,test_generator,y,yhat):

    subset = np.array([i for i,fname in enumerate(test_generator.
   filenames) if (inversor in fname) and (nc in fname)])
    if len(subset)==0:
        return("no file match")
    m = confusion_matrix(y[subset],yhat[subset])

    return(m)

'''
Returns gobal matrix and a dictionary with pairs (inversor, charge
   level) for each subset of the data
'''
def test_metrics(model,test_generator):

    ytest = test_generator.classes
    yhat = np.argmax(model.predict(test_generator),axis=1)
    print("acc:", np.mean(ytest==yhat))
    matrix = confusion_matrix(ytest,yhat)

    subset_matrices = {}

    for inv,nc in COMBINATIONS:
        subset_matrices[(inv,nc)] = subset_matrix(inv,nc,
   test_generator,ytest,yhat)

    return(matrix,subset_matrices)

'''
Refreshing session for safety
'''
def refresh_and_set_session(num_threads = 16):
    import tensorflow as tf

    tf.keras.backend.clear_session()

    '''
    # Maximum number of threads to use for OpenMP parallel regions.
    os.environ["OMP_NUM_THREADS"] = str(num_threads)
    # Without setting below 2 environment variables, it didn't work
   for me. Thanks to @cjw85
    os.environ["TF_NUM_INTRAOP_THREADS"] = str(num_threads)
```

```
    os.environ["TF_NUM_INTEROP_THREADS"] = str(num_threads)

    tf.config.threading.set_inter_op_parallelism_threads(
        num_threads
    )
    tf.config.threading.set_intra_op_parallelism_threads(
        num_threads
    )
    tf.config.set_soft_device_placement(True)
    '''

'''
Performs a SINGLE repetition, meaning one train_val_test split and 5 starts for the same data partition
'''
def main():

    print("rep",rep_name)
    prepare_data()
    split_folders()

    best_model = None
    best_acc = 0.001

    train_generator,validation_generator,test_generator = get_generators(

    splitted_folder="keras_data",

    augmentation=True,

    max_aug_lines=8,

    im_size = (224,224))
    for trial in range(5):
        print("trial",trial)
        refresh_and_set_session()

        model = myModel()

        limit=152
        # Freezing the first _LIMIT_ layers
        for layer in model.layers[:limit]:
            layer.trainable = False
```

```
        for layer in model.layers[limit:]:
            layer.trainable = True

        train_stage(model,
                    train_generator,
                    validation_generator,
                    epochs=1000,
                    lr=0.001,
                    verbose=1)

        trial_acc = model.evaluate(validation_generator)[-1]
        print("val acc",trial_acc)
        if trial_acc > best_acc:
            best_acc = trial_acc
            best_model = model

    global_matrix,subsets = test_metrics(best_model,test_generator)

    pkl.dump(global_matrix,open(f"results/rep_{rep_name}_glbl.pkl", "
    wb"))
    pkl.dump(subsets      ,open(f"results/rep_{rep_name}_sbst.pkl", "
    wb"))

    print(global_matrix)

main()
```

A.2. utils.py

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D,
    Dropout
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.mobilenet import preprocess_input

import pandas as pd
import numpy as np
from PIL import Image
import os
import shutil
import splitfolders



'''
Loads and crops images as specified by right_limit
- right_limit = 220 for transitory state, 500 for transitory+
    stationary state
'''
def processImages(folder,right_limit=220):

    left = 75
    top = 30
    right = right_limit
    bottom = 370
    middle = int((top+bottom)/2)

    spectrograms = os.listdir(folder)

    imgs = [Image.open(folder+"/"+s) for s in spectrograms]
    imgs = [img.crop((left, top, right, bottom)) for img in imgs]

    return([spectrograms,imgs])


'''
```

```
Crawls the og_data directory, processes images and stores them under dest folder
'''
def prepare_data( root = "og_data", dest = "data"):

    try:
        shutil.rmtree("./keras_data")
        shutil.rmtree("./data")
        shutil.rmtree("./models")
    except:
        print("folders not found")

    os.mkdir("keras_data")
    os.mkdir("models")


    os.mkdir(dest)

    for folder in os.listdir(root):

            route = root+"/"+folder+"/"
            names,proc_imgs = processImages(route)
            os.mkdir(dest+"/"+folder)

            for name,img in zip(names,proc_imgs):

                dest_route = dest+"/"+folder+"/"+name
                img.save(dest_route)


'''
Splits folders as specified by ratio in train-val-test
'''
def split_folders(ratio = (.65, 0.15, 0.2)):
    s = np.random.randint(1234567)
    splitfolders.ratio("data", output="keras_data", ratio=ratio,seed=s, group_prefix=None) # default values


'''
Custom spectrogram data augmentation

- Random number of lines betwen 0 and max_lines with a random width between 0 and line_width)
```

```
'''
def aug_preprocess_input(img,max_lines=6,line_width = 20):


    img = preprocess_input(img)

    w = img.shape[0]
    h = img.shape[1]

    nlines = np.random.randint(0,max_lines)
    axes = np.random.uniform(0,1,nlines)>0.5


    for x_line in axes:

        if x_line:

            x0 = np.random.randint(0,w-line_width-1)
            x1 = np.random.randint(x0,x0+line_width)
            img[x0:x1,:] = [0,0,0]

        else:

            y0 = np.random.randint(0,h-line_width-1)
            y1 = np.random.randint(y0,y0+line_width)
            img[:,y0:y1] = [0,0,0]

    return(img)

'''
Obtain train_val_test generators
'''
def get_generators(splitted_folder="keras_data",
                   augmentation=True,
                   batch_size=8,
                   im_size=(256,256),
                   max_aug_lines=6):

    if augmentation:
        aug_preprocess = lambda x : aug_preprocess_input(x,
   max_aug_lines)
        train_preprocess = aug_preprocess
    else:
        train_preprocess = preprocess_input
```

```
124
125
126     train_datagen = ImageDataGenerator(preprocessing_function=
    train_preprocess)
127     test_datagen = ImageDataGenerator(preprocessing_function=
    preprocess_input)
128
129     train_generator = train_datagen.flow_from_directory(
    splitted_folder+'/train',
130                                                         batch_size=
    batch_size,
131                                                         target_size=im_size,
132                                                         class_mode='
    categorical')
133
134     validation_generator = test_datagen.flow_from_directory(
    splitted_folder+'/val',
135                                                             batch_size=8,
136                                                             target_size=
    im_size,
137                                                             class_mode='
    categorical')
138
139     test_generator = test_datagen.flow_from_directory(splitted_folder+
    '/test',
140                                                       batch_size = 300,
141                                                       target_size=im_size,
142                                                       class_mode='
    categorical',
143                                                       shuffle = False)
144
145     return(train_generator,validation_generator,test_generator)
```

A.3. Interpretación LIME

```
from tensorflow.keras.models import load_model
from utils import get_generators
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import warnings
import os
import tensorflow

import matplotlib.pyplot as plt
from skimage.segmentation import mark_boundaries

import sys

try:
    import lime
except:
    sys.path.append(os.path.join('..', '..'))
    import lime
from lime import lime_image


# Gets all the test set hits (correctly classified observations) interpretations
# inv must be one of AB1,ABB1,TM1
# R must be an integer between 1 and 5 (included) representing the wearing status
def get_inv_expls(inv,R):

    c = R-1

    # Load Model and train,val and test
    model = load_model('model_interpretability.h5')

    train_generator,validation_generator,test_generator = get_generators(splitted_folder="keras_data",

          augmentation=True,

          max_aug_lines=8,
```

```
37          im_size = (224,224))
38
39
40      # From test generator, get filenames (which contains the inversor)
        and the images and wearing statuses themselves
41      filenames = test_generator.filenames
42      x_test, y_test = test_generator.next()
43
44
45      # correct_c represents all the files in test that are correctly
    predicted and correspond to the desired inversor 'inv'
46      yhat = np.argmax(model.predict(test_generator),axis=-1)
47      ytest = test_generator.labels
48
49      yhat_is_c = yhat == c
50      ytest_is_c = ytest == c
51
52      filename_is_inv = np.array([inv in f for f in filenames])
53
54      correct_c = np.logical_and(yhat_is_c,ytest_is_c)
55
56      correct_c = np.logical_and(correct_c,filename_is_inv)
57      correct_c = np.where(correct_c)[0]
58
59
60
61      def explain_instance(x,label,plot=False, pos = True):
62          explainer = lime_image.LimeImageExplainer()
63          # Hide color is the color for a superpixel turned OFF.
    Alternatively, if it is NONE, the superpixel will be replaced by
    the average of its pixels
64          explanation = explainer.explain_instance(x_test[0].astype('
    double'), model.predict, top_labels=5, hide_color=0, num_samples
    =1000)
65
66          tempf, maskf = explanation.get_image_and_mask(label,
    positive_only=pos,negative_only=not pos, num_features=5, hide_rest
    =False)
67          tempt, maskt = explanation.get_image_and_mask(label,
    positive_only=pos,negative_only=not pos,num_features=5, hide_rest=
    True)
68          if plot:
69              plt.imshow(mark_boundaries(tempf / 2 + 0.5, maskf))
```

```
        imf = mark_boundaries(tempf / 2 + 0.5, maskf)
        imt = mark_boundaries(tempt / 2 + 0.5, maskt)
        return([imf,imt])


    # for each instance, explain its clasification
    exps_c = [explain_instance(x_test[i],c,pos = True) for i in
    correct_c]


    #Each block represents a different format

    #format 0
    m = len(exps_c)
    x = sum([exps_c[i][1] for i in range(m)])/m
    plt.imshow(x)
    plt.imsave(f"interprets/{inv}_R{R}_f0.jpg",x)

    #format 1
    not_gray = lambda x : np.expand_dims((x != [0.5,0.5,0.5]).all(-1),
    axis=-1)
    ximp = (sum([not_gray(exps_c[i][1]) for i in range(m)]))/m
    x = sum([x_test[i] for i in correct_c])/m
    plt.imshow(ximp)
    plt.imsave(f"interprets/{inv}_R{R}_f1.jpg",ximp.squeeze())

    #format 2
    xmask = (ximp<=0)*1.0
    plt.imshow(xmask)
    plt.imsave(f"interprets/{inv}_R{R}_f2.jpg",xmask.squeeze())

    #format 3
    xmap = (x-x.min())
    xmap = xmap/xmap.max()

    plt.imshow(xmap)
    plt.imsave(f"interprets/{inv}_R{R}_f3.jpg",xmap)


    #format 4
    x[np.repeat(xmask==1,3,axis=-1)]=0
    xgrad = x*ximp

    xgrad = xgrad-xgrad.min()
```

```
    xgrad = xgrad/xgrad.max()
    plt.imshow(xgrad)
    plt.imsave(f"interprets/{inv}_R{R}_f4.jpg",xgrad)
```

5.4. Grad-CAM

```
import tensorflow as tf

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import shap
import warnings
import os
import eli5
from PIL import Image

from tensorflow.keras.models import load_model
from tensorflow.keras.utils import plot_model

from utils import get_generators

import matplotlib.pyplot as plt
import matplotlib.cm


def gradcam(R):
    R = 5
    C = R-1

    tf.compat.v1.disable_eager_execution()

    model = load_model('model_interpretability.h5')

    dont_use_softmax = True

    if dont_use_softmax:
        l = model.get_layer(index=-1) # get the last (output) layer
        l.activation = tf.keras.activations.linear # swap activation



    train_generator,validation_generator,test_generator =
   get_generators(splitted_folder="keras_data",

          augmentation=True,

```

```
        max_aug_lines=8,

        im_size = (224,224))

    x_train, y_train = train_generator.next()
    x_test, y_test = test_generator.next()


    model.layers[0].input.set_shape((None,)+x_test[0].shape)


    def explain_prediction(x,target_class,plot=False):
        print("Computing Explanation...")
        x = np.expand_dims(x,0)
        expl = eli5.explain_prediction(model, x,targets=[target_class
    ])

        if plot:
            I = eli5.format_as_image(expl, alpha_limit=1.0, colormap=
    matplotlib.cm.Greys)
            display(I)
        return(expl)


    yhat = np.argmax(model.predict(x_test),1)
    ytest = test_generator.labels

    hits = np.where(np.logical_and(yhat==ytest,ytest==C))[0]

    expls = []
    for x in x_test[hits]:
        expls.append(explain_prediction(x,target_class=C))

    heatmaps = []

    for e in expls:
        image = e.image
        heatmap = e.targets[0].heatmap
        heatmap_im = eli5.formatters.image.expand_heatmap(heatmap,
    image, resampling_filter=Image.BOX)
        heatmaps.append(np.array(heatmap_im))
```

```
    count = len(expls)
    avg = heatmaps[0]/count

    for h in heatmaps[1:]:
        avg = avg + h/count


    plt.imshow(avg)


    for e in expls:
        I = eli5.format_as_image(e, alpha_limit=1.0, colormap=
matplotlib.cm.Blues)
        display(I)


    Is = np.array([np.array(eli5.format_as_image(e, alpha_limit=1.0,
colormap=matplotlib.cm.Greys)) for e in expls])

    plt.imsave(f"gradcam_R{R}.jpg",np.mean(Is,axis=0)/255)
```

Índice de figuras

Índice de tablas