TÍTULO

PREDICCIÓN DE PÉRDIDA DE POTENCIA EN PANELES FOTOVOLTAICOS MEDIANTE EL ANÁLISIS DE IMÁGENES

AUTOR

José Alberto Rodríguez Álvarez

Esta edición electrónica ha sido realizada en 2023

Tutores	D. Diego Marín Santos; D. Manuel E. Gegúndez Arias
Instituciones	Universidad Internacional de Andalucía
Curso	*Máster en Big Data (2021-2022)*
©	José Alberto Rodríguez Álvarez
©	De esta edición: Universidad Internacional de Andalucía
Fecha documento	2022

Trabajo Fin de Máster Big Data

Predicción de pérdida de potencia en paneles fotovoltaicos mediante el análisis de imágenes.

Autor: José Alberto Rodríguez Álvarez

Tutores: Diego Marín Santos

Manuel Emilio Gegundez Arias

Enero de 2023

AGRADECIMIENTOS

Tras un largo período de esfuerzo y constancia, finalizo mis estudios en el Máster de Big Data impartido por la UNIA.

Quería agradecer el apoyo de mis padres, hermana y abuelos. También, agradecer a los profesores de la Universidad Internacional de Andalucía, y, en especial, a mis tutores, Diego Marín Santos y Manuel Emilio Gegundez Arias, que me han guiado y ayudado para poder abordar este proyecto.

RESUMEN

La inspección de paneles fotovoltaicos en busca de desperfectos que comprometan el rendimiento de los mismos es una tarea que, generalmente, se realiza de manera visual por los operarios encargados del mantenimiento de las instalaciones. Esta técnica supone una gran carga de trabajo en instalaciones grandes y requiere de una especialización para detectar correctamente los problemas.

El objetivo de este proyecto es el análisis de imágenes de paneles fotovoltaicos mediante un sistema basado en técnicas de Big Data y Deep Learning para predecir las posibles pérdidas energéticas ocasionadas por la acumulación de suciedad en la superficie del panel analizado.

Para abordar el proyecto, se ha diseñado una red neuronal convolucional que nos permita resolver el problema de regresión planteado. Además, como comparación, vamos a usar la red VGG16 para adaptarla a nuestro problema.

La experimentación se ha llevado a cabo en un conjunto de 45754 imágenes de paneles fotovoltaicos en la que, cada imagen, tiene la información de la pérdida de potencia debido a la acumulación de suciedad. El entrenamiento, validación y evaluación de la red se realizó sobre el 80%, 10% y 10% del total de imágenes, respectivamente.

El error absoluto medio obtenido en la red neuronal convolucional diseñada ha sido 3.729, mientras que el error absoluto medio obtenido en la red VGG16 ha sido 3.754, resultados bastante similares entre ambos modelos.

Por lo tanto, teniendo en cuenta los resultados obtenidos, se podría afirmar que tanto la red CNN propuesta como la red VGG16 podrían utilizarse en la tarea del análisis de paneles fotovoltaicos para detectar aquellos paneles en los que la acumulación de suciedad pueda provocar una pérdida en el rendimiento.

ABSTRACT

The inspection of photovoltaic panels in search of defects that compromise their performance is a task generally carried out visually by the operators in charge of the facilities's maintenance.This technique is a heavy workload in large installations and requires specialisation to correctly detect problems.

The objetive of this proyect is the analysis of photovoltaic solar panels' images through a system based on Big Data and Deep Learning techniques to predict possible energy losses caused by the accumulation of dirt on the surface of the analyzed panel.

In order to achieve this aim, a convolutional neural network that allows us to solve the regression problema has been designed. Also, as a comparison, we are going to use the VGG16 network to addapt it to our problem.

The experimentation has been carried out on a set of 45754 images of photovoltaic panels, in which each image has the information of the power loss due to dirt accumulation. The training, validation and evaluation of the network has been performed on 80%, 10% and 10% of the total images, respectively.

The mean absolute error obtained in the designed convolutional neural network was 3.729, whereas the mean absolute error obtained in the VGG16 network was 3.754. As it can be observed, both models are very similar.

Therefore, taking into account the results obtained, it could be said that both models could be used in the task of analyzing photovoltaic panels to detect those in which the dirt accumulation could induce a loss in performance.

Tabla de contenido

1 Propuesta de TFM.

Cuando una fuente de energía se basa en el aprovechamiento de un recurso natural inagotable como el sol, el viento, el agua o la biomasa, se dice que esa fuente de energía es renovable. La energía renovable se caracteriza por no utilizar combustibles fósiles, sino que utiliza recursos naturales renovables. Al no utilizar combustibles fósiles, no se producen gases de efecto invernadero y otras emisiones contaminantes, causantes del cambio climático.

Existen varios tipos de energías renovables, como:

- Energía eólica: es la energía obtenida del viento.
- Energía solar: es la energía obtenida del sol. Las principales tecnologías son la solar fotovoltaica, en la que se aprovecha la luz del sol, y la solar térmica, en la que se aprovecha el calor del sol.
- Energía hidráulica o hidroeléctrica: es energía obtenida de los ríos y arroyos de agua dulce
- Biomasa y biogás: es la energía que se extrae de materia orgánica.
- Energía geotérmica: es la energía calorífica contenida en el interior de la Tierra.
- Energía mareomotriz: es la energía obtenida de las mareas.

En este proyecto nos centraremos en la energía solar fotovoltaica. La energía solar fotovoltaica es aquella que se obtiene al transformar la luz solar en electricidad empleando una tecnología basada en el efecto fotoeléctrico, mediante el uso de paneles fotovoltaicos. El efecto fotoeléctrico se produce cuando una superficie se expone a un tipo de radiación electromagnética. En este momento, la superficie absorbe la luz incidente y emite electrones. En el caso particular de los paneles fotovoltaicos, de la radiación solar se absorbe una partícula llamada fotón que reacciona con los electrones del material de la célula fotovoltaica. Esta reacción se verá afectada si el panel presenta algún tipo de desperfecto, como una capa de suciedad en la superficie del panel.

El objetivo de este trabajo es crear un sistema que reciba una fotografía de un panel fotovoltaico y de una predicción de la pérdida de potencia en la transformación de energía solar en energía eléctrica según el nivel de suciedad presente en la superficie del panel.

1.1 Motivación del TFM.

En este apartado se tratará la motivación de este TFM. Para entender la motivación, previamente comentaré mi actividad profesional. Actualmente, trabajo en el sector de las energías renovables, concretamente en el desarrollo de sistemas SCADA. Un SCADA es un sistema que se utiliza para la supervisión, control y adquisición de datos. El sistema consta de componentes software y hardware que proporcionan una plataforma intuitiva de visualización y análisis en tiempo real a través de una interfaz gráfica.

Podemos destacar las siguientes características de estos sistemas:

- Monitorización en tiempo real: se permite visualizar en tiempo real las medidas y valores que ofrecen los dispositivos de una planta renovable.
- Control sobre dispositivos de campo: se permite ejecutar órdenes desde el sistema y actuar sobre los dispositivos y elementos de una instalación operados bajo control remoto.
- Notificación de alertas: se puede recibir de forma automática avisos de fallo en dispositivos, problemas de comunicación, disparos en subestaciones eléctricas o baja producción.
- Explotación de datos mediante informes: se puede comparar datos históricos para analizar tendencias de producción y examinar el rendimiento de equipos. También, se puede generar gráficas con datos en tiempo real cruzando múltiples señales, crucial para descubrir patrones de comportamiento e indisponibilidad de sus equipos, lo que permite adoptar medidas para impulsar la productividad.

Por otra parte, aprovechando que finalizo el máster en Big Data, quería aprovechar la oportunidad de poder aplicar todo lo aprendido en las diferentes asignaturas para abordar el problema planteado en el apartado anterior. En concreto, el objetivo de este TFM es usar los conocimientos adquiridos en las asignaturas de Big Data y Deep Learning para crear un sistema que reciba una fotografía de un panel fotovoltaico y de una predicción de la pérdida de potencia en la transformación de energía solar en energía eléctrica según el nivel de suciedad presente en la superficie del panel. De esta manera, podría aplicar los conocimientos adquiridos en el TFM con un caso práctico cercano a mi ámbito de trabajo.

1.2 Objetivos generales y específicos.

Dentro de los objetivos generales de este TFM, incluimos la tarea de aplicar los conocimientos aprendidos en asignaturas enfocadas al Big Data para procesar los datos con los que vamos a abordar el proyecto y obtener información relevante de ellos. Además, como objetivos generales, también incluimos la tarea de aplicar los conocimientos aprendidos en asignaturas enfocadas al Machine Learning y Deep Learning para diseñar una red neuronal que sea capaz de tomar una imagen como entrada y dar un valor numérico como salida, siendo la salida la predicción de la red neuronal.

Una vez conseguimos los objetivos generales, abordaríamos los objetivos específicos. Dentro de los objetivos específicos, queremos que nuestro modelo de red neuronal sea capaz de analizar una imagen de un panel fotovoltaico y que el valor numérico que se obtenga al final no sea un valor cualquiera, sino que sea un valor coherente con respecto al nivel de suciedad de la superficie del panel.

1.3 Estructura del documento.

El documento está estructurado de la siguiente manera.

En primer lugar, en el documento se empieza con los agradecimientos a las personas que me han ayudado o apoyado durante el tiempo dedicado a este Máster.

Seguidamente, se hace un breve resumen del proyecto, tanto en español como en inglés.

Continuamos con la propuesta de TFM, y los objetivos generales y específicos que se desean conseguir en el proyecto.

A continuación, se incluyen una serie de trabajos que han sido de apoyo para abordar el proyecto o trabajos que han servido para aprender conceptos nuevos, conceptos que se aplicarán a este TFM.

En los siguientes apartados, se describe la base de datos que se va a emplear en el proyecto, así como el planteamiento metodológico, donde se detallan los modelos de redes usados en el proyecto y el entrenamiento de dichas redes.

Por último, se exponen los resultados conseguidos con los modelos de redes usados y se finaliza el documento con las conclusiones, valoraciones y futuros trabajos por realizar.

2 Introducción. Estado del arte.

Anteriormente, se ha comentado brevemente el funcionamiento de un panel fotovoltaico. Vamos a profundizar en esto.

Un panel fotovoltaico está compuesto por diferentes materiales, como encapsulantes, caja de conexiones, una capa de vidrio o una capa de celdas fotovoltaicas, entre otros.

Figura 1

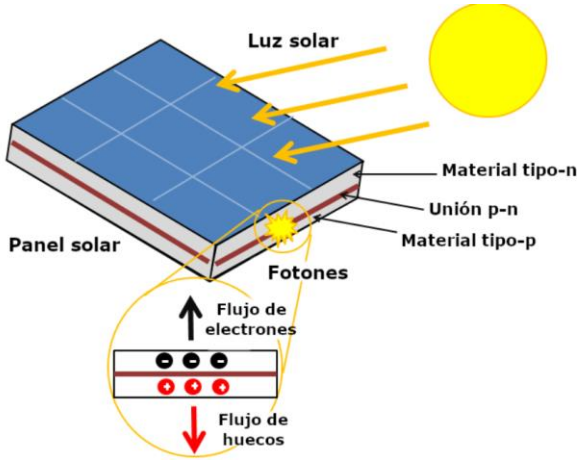
Panel fotovoltaico: Capas

Panel Fotovoltaico. (2022, 30 agosto). APREAN. https://www.aprean.com/energia-fotovoltaica/panel-fotovoltaico/

La capa de celdas fotovoltaicas no son más que un conjunto de celdas fotovoltaicas conectadas entre sí. Una celda fotovoltaica consta de dos tipos diferentes de semiconductores, de tipo p y de tipo n, conectados entre sí para formar una unión p-n, creando de esta manera un campo eléctrico. Este campo hace que las partículas cargadas negativamente se muevan en una dirección y las cargadas positivamente en la dirección opuesta. Cuando la luz solar, compuesta por fotones, incide sobre la superficie de la celda fotovoltaica, la energía del fotón se transfiere a un electrón del material semiconductor. En este momento, los electrones se pueden mover por el material, creando una corriente eléctrica en la célula a causa del movimiento de los electrones.

Figura 2

Efecto fotovoltaico



Célula fotovoltaica Enciclopedia de Energia. (s. f.).
https://energyeducation.ca/Enciclopedia_de_Energia/index.php/Célula_fotovoltaica

Como se ha comentado previamente, el panel fotovoltaico necesita luz solar para producirse el efecto fotoeléctrico. Si un panel presenta defectos en la superficie, como acumulación de nieve, excrementos de aves o acumulación de polvo, este efecto fotoeléctrico se verá disminuido, ya que la célula solar recibirá luz solar con menor intensidad.

Consultando bibliografía que apoye lo anteriormente descrito, encontramos un artículo (Mohammad Reza Maghami, Hashim Hizam, Chandima Gomes, Mohd Amran Radzi, Mohammad Ismael Rezadad, Shahrooz Hajighorbani, 2016) que trata sobre la pérdida de potencia en paneles fotovoltaicos debido al factor ambiental de la suciedad. En el artículo se detalla que la potencia de salida entregada por un panel fotovoltaico depende de la cantidad de radiación que llega a las células solares. Se puede encontrar diversos factores que determinan el rendimiento de un módulo fotovoltaico. Entre ellos, el medio ambiente es un factor que influye directamente en el rendimiento fotovoltaico. Encontramos que las pérdidas por suciedad se dan cuando se tiene una pérdida de potencia resultante de la acumulación de nieve, la suciedad, el polvo y otras partículas que cubren la superficie del módulo fotovoltaico. Los autores citan varios trabajos en los que se estudió el efecto de la acumulación de polvo en el rendimiento del panel fotovoltaico, entre los que destacamos:

- Hottel y Woertz, en el año 1942, realizaron una prueba en un área industrial cerca de de Boston, Massachusetts, para investigar el efecto de la acumulación de polvo en los paneles solares. Detectaron que, debido al polvo que se había acumulado en la superficie del panel solar, se produjo una pérdida promedio del 1 % de la radiación solar incidente en la superficie del panel.
- En el 2001, Kimber et al., estudió los efectos de la suciedad en paneles fotovoltaicos en California, Estados Unidos. El estudio estaba destinado a crear un modelo destinado a predecir las pérdidas anuales por suciedad con mayor precisión ya que, hasta la fecha, se tomaba un valor constante para las pérdidas por suciedad. Los resultados del estudio fueron que las pérdidas anuales provocadas por la suciedad oscilaban entre el 1,5 % y el 6,2 %, según la ubicación de la planta fotovoltaica.

Una vez validado nuestro planteamiento del problema con otros artículos y estudios de diferentes autores, lo siguiente será describir varios trabajos en los que se busque un objetivo similar al nuestro, que se trata de usar técnicas de Big Data y Deep Learning para crear un sistema que reciba una fotografía de un panel fotovoltaico y de una predicción de la pérdida de potencia.

2.1 Deep Solar Eye.

El primer trabajo que encontramos es el proyecto *Deep Solar Eye* (Sachin Mehta, Amar P. Azad, Saneem A. Chemmengath, Vikas Raykar, Shivkumar Kalyanraman, 2018). En este proyecto, los autores buscan desarrollar un modelo usando técnicas de Deep Learning para predecir simultáneamente la pérdida de potencia provocada por la acumulación de suciedad, la localización de la suciedad y la categoría de la suciedad, dada una imagen RGB de un panel solar.

2.1.1 Deep Solar Eye: planteamiento metodológico.

Los autores del proyecto exponen la problemática que supone la presencia de defectos o la acumulación de suciedad en la superficie del panel fotovoltaico. La construcción de granjas solares, que no es más que un recinto donde se concentran instalaciones de paneles fotovoltaicas, es cada vez más popular. Esto hace que sean imprescindibles tareas de mantenimiento en la limpieza de la suciedad presente en la superficie de paneles fotovoltaicos para no tener una pérdida en el rendimiento del mismo. Para llevar a cabo estas tareas de mantenimiento, es necesario conocer el tipo de suciedad presente en el panel, ya que las acciones correctivas no serán iguales si el panel está cubierto por una capa de polvo, o si el panel está cubierto de excrementos de aves. El tipo de suciedad o defecto presente en el panel se puede reconocer fácilmente observando la imagen del panel fotovoltaico. Sin embargo, para analizar el impacto de la suciedad o el defecto en el rendimiento del panel solar, es necesario conocer detalles, como la cantidad de suciedad presente, el tipo y la ubicación de la suciedad en el panel. Esta información será útil tanto para estimar el impacto en el rendimiento del panel, como para decidir qué tipo de acción correctiva se debe realizar, aspectos fundamentales para el mantenimiento eficiente de la granja solar. Por este motivo, en el proyecto los autores buscan predecir la pérdida de potencia provocada por la acumulación de suciedad, la localización y categoría de la misma.

Para abordar el problema planteado, los autores definen un sistema basado en redes neuronales convolucionales. En concreto, se define un enfoque de varios pasos que permite el entrenamiento del sistema con etiquetado automático de datos. Los pasos son:

- En la primera etapa se entrena una red de clasificación basada en redes neuronales convolucionales, a la que los autores nombran ImpactNet, para predecir la pérdida de potencia. En este caso, se aborda como un problema de clasificación, ya que la predicción de pérdida de potencia se da dentro de un intervalo perteneciente a una categoría previamente definida.
- En la segunda etapa se determina la zona donde se concentra la capa de suciedad, también usando redes neuronales convolucionales.
- Finalmente, se predice el tipo de suciedad presente en la superficie del panel fotovoltaico, pudiendo ser polvo, excrementos de aves o nieve, entre otros.

2.1.2 Deep Solar Eye: datos.

El principal problema de proyectos del tipo de *Deep Solar Eye* se debe a la escasez de conjuntos de datos con los que poder abordar el problema. Por este motivo, los autores del estudio crearon un dataset con los que poder entrenar las redes neuronales convoluciones diseñadas.

Para crear el dataset, pusieron dos paneles fotovoltaicos, uno al lado del otro, siendo los dos paneles de las mismas características. El objetivo era llenar la superficie de un panel con diferente tipo y espesor de suciedad, mientras que el otro panel se mantenía limpio. Además, frente a los paneles fotovoltaicos, pusieron una cámara para capturar imágenes RGB con las que poder entrenar las redes del modelo. Finalmente, registraron la energía que exportaba cada uno de los paneles. De esta manera, se puede obtener la pérdida de potencia del panel que presenta suciedad con respecto al panel de referencia, que es el que estaba completamente limpio.

2.1.3 Deep Solar Eye: resultados y conclusiones.

En el artículo se especifica que, durante el entrenamiento, se prueban diferentes técnicas para encontrar el mejor modelo, como el uso de data augmentation, el uso de otra variable más de entrada en el modelo (la variable irradiancia), o cambios en la estructura del modelo.

Una vez que encontraron el mejor modelo, los autores del proyecto recogieron datos nuevos durante tres semanas para poder comprobar el desempeño del modelo con datos nuevos. El modelo consiguió una precisión del 86.5%.

Por último, como conclusión, en el artículo se especifica que llevaron a cabo un último experimento. En este último experimento, recopilaron imágenes de Internet de paneles solares con defectos que no estaban presentes en el dataset con el que el modelo fue entrenado, como el uso de imágenes donde el panel presentaba acumulación de nieve en la superficie. Recordemos que una parte del diseño de la red convolucional estaba destinado a predecir el tipo suciedad. Debido a esta característica y debido a que no se disponía del dato de pérdida de potencia en las imágenes nuevas, los autores se limitaron a predecir el tipo de suciedad. En esta tarea, los autores comentan que la red neuronal consiguió una precisión del 87%, lo que quiere decir que la red diseñada generaliza bien.

El proyecto *Deep Solar Eye* persigue un objetivo similar al nuestro. Se pretende desarrollar un modelo usando técnicas de Deep Learning para predecir simultáneamente la pérdida de potencia provocada por la acumulación de suciedad, la localización y categoría de la misma, dada una imagen RGB de un panel solar. En el proyecto, la pérdida de potencia se da dentro de un intervalo perteneciente a una categoría previamente definida, que hace que el problema tenga un enfoque de clasificación. Sin embargo, en nuestro proyecto, aunque el objetivo sigue siendo desarrollar un modelo usando técnicas de Deep Learning para predecir la pérdida de potencia provocada por la acumulación de suciedad, queremos que la predicción sea el valor numérico, no queremos incluirlo dentro de una categoría, lo que hace que el problema tenga un enfoque de regresión. Debido a esto y debido a que no se han encontrado más trabajos relacionados con la temática de este TFM, vamos buscar proyectos que se basen en redes convolucionales para hacer regresión a partir de una imagen de entrada.

2.2 House Prices.

Como hemos comentado previamente, el proyecto *House Prices* (Markus Rosenfelder, 2022) no está relacionado con la temática de este TFM, pero nos servirá para aprender a adaptar un problema de clasificación a un problema de regresión a partir de una imagen de entrada, usando redes neuronales convolucionales creadas por nosotros mismos y, también, usando redes neuronales convolucionales ya creadas y entrenadas por otros autores, lo que se conoce como transfer learning. Para ello, el autor del proyecto dispone de un conjunto de imágenes de casas de Nueva York y un fichero CSV en el que se recoge información de cada casa, como la zona donde está ubicada, el número de baños, número de habitaciones o superficie que ocupa, además del precio de la vivienda. Esta información se usará para, una vez desarrollado el modelo de redes neuronales convolucionales, predecir el precio de las casas.

2.2.1 House Prices: planteamiento metodológico.

El autor de este proyecto expone la problemática que supone encontrar tutoriales en Internet que desarrollen problemas de regresión usando redes neuronales convolucionales complejas con imágenes como entrada de datos, ya que la mayoría de los tutoriales usan redes neuronales convolucionales sencillas, o usan la técnica de transfer learning o, simplemente están enfocados a problemas de clasificación. Por este motivo, el objetivo es crear un proyecto que aborde todos estos aspectos, usando el conjunto de datos de las casas de Nueva York para predecir el precio de la vivienda.

El proyecto consta de los siguientes pasos:

- Usar ImageDataGenerators y Pandas DataFrames para cargar los datos en el sistema.
- Explicar técnicas de data augmentation para intentar mejorar los resultados.
- Crear una red neuronal convolucional destinada a regresión.
- Adaptar la red EfficientNet a un problema de regresión, para usar la técnica de transfer learning.
- Comparar los resultados de la red creada por el autor con los resultados de la red EfficientNet.

2.2.2 House Prices: datos.

Como hemos comentado previamente, para el desarrollo de este proyecto, el autor ha usado un conjunto de imágenes de casas de Nueva York y un fichero CSV en el que se recoge información de cada casa, como la zona donde está ubicada, el número de baños, número de habitaciones o superficie que ocupa, además del precio de la vivienda.

Con respecto a las imágenes, el autor explica que ha aplicado técnicas de data augmentation para entrenar las redes neuronales convolucionales, tanto la red creada por el mismo, como la red EfficientNet. Entre las técnicas de data augmentation usadas, podemos destacar las siguientes: reescalado, rotación, aplicación de zoom, voltear horizontalmente las imágenes o aumentar el brillo. El objetivo de estas técnicas es que, durante la fase de entrenamiento, nuestro modelo no tenga exactamente la misma imagen en las diferentes epochs. Con esto se busca que el modelo se exponga a más aspectos de los datos, aprenda más patrones y generalice mejor.

Con respecto a los datos contenidos en el fichero CSV, el autor especifica que solo se va a hacer uso de la columna precio. Aunque el fichero contiene más información, como la zona donde está ubicada, el número de baños, número de habitaciones o superficie que ocupa, las imágenes de las casas son exteriores, por lo que no tendría utilidad aumentar la complejidad usando el resto de variables en el sistema.

2.2.3 House Prices: resultados y conclusiones.

Al final del proyecto, el autor compara los resultados obtenidos al entrenar la red neuronal convolucional diseñada por él mismo con los resultados obtenidos al entrenar la red EfficientNet. A la vista de los resultados mostrados, EfficientNet alcanza su error de validación más bajo en la cuarta época, mientras que la red creada por el autor necesita 18 épocas para llegar a su mínimo. Otro dato interesante que comparte es que, la red propuesta, tarda 17 minutos en alcanzar el valor más bajo de error, mientras que EfficientNet sólo tarda 3 minutos. También, comenta que el error cometido por la red propuesta es del 27.8%, mientras que el error cometido por EfficientNet es del 23.9%.

Como conclusión, el autor concluye el proyecto argumentando la idea de que, con el uso de transfer learning, se puede disminuir el tiempo de entrenamiento y, al mismo tiempo, disminuir el error cometido en la predicción.

2.3 Book Prices

El siguiente proyecto que hemos encontrado es *Book Prices* (Theethat Anuraksoontorn, 2021). De nuevo, este proyecto no está relacionado con la temática de este TFM, pero nos servirá para aprender a crear redes neuronales convolucionales aplicadas a regresión donde los datos de entrada son imágenes. El objetivo de este proyecto es crear un sistema basado en Deep Learning usando redes neuronales convolucionales para predecir el precio de un libro analizando una imagen de la portada del mismo. Para ello, el autor del proyecto dispone de un conjunto de imágenes de libros y un fichero CSV, en el que se recoge información relativa al autor, nombre del libro, isbn o precio, entre otras variables.

2.3.1 Book Prices: planteamiento metodológico.

El autor de este proyecto, al igual que en el proyecto analizado anteriormente, expone la problemática que supone encontrar tutoriales en Internet que desarrollen problemas de regresión donde los datos de entrada sean imágenes. Debido a este inconveniente, el autor aborda la tarea de crear un sistema usando redes neuronales convolucionales con el objetivo de predecir el precio de un libro analizando la imagen de la portada del mismo.

El proyecto consta de los siguientes pasos:

- Cargar los datos en el sistema para analizarlos.
- Preprocesamiento de imágenes.
- Crear la red neuronal convolucional.
- Analizar los resultados obtenidos.

2.3.2 Book Prices: datos.

Como hemos comentado previamente, para el desarrollo de este proyecto, el autor ha usado un conjunto de imágenes de portadas de libros y un fichero CSV en el que se recoge información relativa al autor, nombre del libro, isbn o precio.

Con respecto a las imágenes, el autor detalla el preprocesamiento que lleva a cabo antes de alimentar la red neuronal con estos datos. En concreto, las operaciones que hace sobre las imágenes son:

- Conversión a escala de grises: el principal objetivo de convertir las imágenes RGB a escala de grises es reducir los requisitos computacionales.
- Recorte de imágenes: el objetivo de recortar imágenes es eliminar regiones no deseadas. En el caso de este proyecto, la esquina de la portada del libro no influye en el precio de su libro.
- Reescalado: el objetivo vuelve a ser reducir los requisitos computacionales.

Con respecto a los datos contenidos en el fichero CSV, el autor especifica que solo se va a hacer uso de la columna precio. Aunque el fichero contiene más información como el nombre del autor, nombre del libro, o isbn, no van a estar en el entrenamiento de la red neuronal.

2.3.3 Book Prices: resultados y conclusiones.

El autor crea una red neuronal convolucional no demasiado compleja para esta tarea. El autor del proyecto grafica la curva correspondiente a las pérdidas obtenidas en la fase de entrenamiento y otra curva correspondiente a las pérdidas obtenidas en la fase de validación.

A la vista de los resultados obtenidos, el autor comenta que el modelo está sufriendo sobre aprendizaje. Esto quiere decir que el modelo se ha aprendido los datos de entrenamiento y no es capaz de generalizar ante datos nuevos. Por este motivo, y como conclusión del proyecto, el autor detalla una serie de pasos para abordar el problema de sobre aprendizaje, que son:

- Añadir más datos.
- Crear una estructura de red más compleja.
- Usar técnicas de data augmentation.

3 Materiales.

En este apartado, se explicará la base de datos elegida para abordar el proyecto.

3.1 Base de datos.

Recordemos que el objetivo de este proyecto es desarrollar un modelo usando técnicas de Deep Learning para predecir la pérdida de potencia provocada por la acumulación de suciedad. Para abordarlo, necesitamos una base de datos que contenga imágenes de paneles fotovoltaicos y, además, necesitamos que cada una de las imágenes contenga la información del valor de pérdida de potencia provocado por la acumulación de suciedad en la superficie del panel.

Una base de datos de estas características es difícil de encontrar actualmente por lo que, aprovechando que la base de datos analizada en el proyecto *Deep Solar Eye* (Sachin Mehta, Amar P. Azad, Saneem A. Chemmengath, Vikas Raykar, Shivkumar Kalyanraman, 2018) se puede usar con fines académicos, se hará uso de estos datos.

Como se explicó en el apartado 2.1.2, para crear el dataset, el procedimiento fue situar dos paneles fotovoltaicos, de las mismas características, uno al lado del otro. La superficie de uno de los paneles se ensució con diferente tipo y espesor de suciedad, mientras que el otro panel se mantuvo limpio. Frente a los paneles fotovoltaicos, se puso una cámara para capturar imágenes RGB con las que poder entrenar las redes del modelo. Finalmente, se registró la energía que exportaba cada uno de los paneles para poder obtener la pérdida de potencia del panel que presentaba suciedad con respecto al panel de referencia, que es el que estaba completamente limpio.

El resultado del valor de pérdida de potencia calculada, junto con el valor de irradiancia que hubo en el momento de captura de las imágenes, además de la fecha y hora, lo usaron los autores del dataset para nombrar la imagen. De esta manera, tomando de ejemplo una imagen cualquiera, el nombre sería el siguiente:

solar_Fri_Jun_16_9__48__13_2017_L_0.901234986193_I_0.23148627451.jpg

En la imagen podemos ver la fecha y hora en la que fue capturada la imagen. En este caso, la imagen fue tomada el 16 de junio del 2017, a las 9:48:13 horas.

Por otro lado, el valor situado a la derecha de la letra L sería el valor de pérdida de potencia, que en este caso es 0.9012, mientras que el valor situado a la derecha de la letra I sería el valor de irrandiancia, que en este caso es 0.2314. Estos valores se encuentran recogidos en el rango [0,1].

Si graficamos la imagen anterior, podemos ver que las imágenes son del siguiente estilo:

Figura 3: Imagen panel fotovoltaico

Observando la imagen se puede ver que contiene una fotografía de un panel solar, que puede tener suciedad o estar limpio. En este caso, se puede ver que el panel tiene suciedad que, según los datos facilitados por los autores del dataset, provoca una pérdida de potencia del 90%, si lo pasamos a tanto por ciento, con respecto al panel limpio de referencia.

El dataset está formado por 45754 imágenes como la anterior, cada una con sus valores de pérdida de potencia e irradiancia registrada. Cabe destacar que el dataset no está dividido en conjunto de entrenamiento, validación y test, por lo que, posteriormente, tendremos que abordar esta tarea, ya que es importante tener conjuntos de datos diferentes para entrenar y validar el modelo creado.

3.1.1 Base de datos: análisis de datos.

Una vez que hemos explicado cómo está formado el conjunto de datos que vamos a utilizar en el proyecto, el siguiente paso será hacer un análisis de los datos.

Como se ha comentado previamente, el nombre de cada imagen contiene la fecha y hora en la que fue tomada, además del nivel de pérdida de potencia y el nivel de irradiancia que incidió sobre la superficie en ese momento. Para poder explorar los datos y usarlos posteriormente en la fase de diseño del modelo, se ha creado un dataframe que contiene toda esta información. Un DataFrame es una estructura de datos con dos dimensiones en la cual se puede guardar datos de distintos tipos. Nuestro dataframe contiene las siguientes columnas:

- Image Name: esta columna contiene el nombre completo de la imagen.
- Date: esta columna contiene la fecha y hora en la que fue tomada la imagen, en formato AAAA-MM-DD HH:MM:SS, es decir, año-mes-día hora:minuto:segundo.
- Power Loss: esta columna contiene el valor de pérdida de potencia del panel fotovoltaico sucio con respecto al panel fotovoltaico limpio.
- Irradiance: esta columna contiene el valor de irradiancia que incidió sobre la superficie en el momento de captura de la imagen.

Una vez explicada la estructura de nuestro dataframe, pasamos a visualizar una muestra del mismo:

	Image Name	Date	Power Loss	Irradiance
6818	solar_Tue_Jun_13_9__46__49_2017_L_0.0474843723...	2017-06-13 09:46:49	4.748	29.673
35595	solar_Tue_Jun_13_9__46__54_2017_L_0.0273312333...	2017-06-13 09:46:54	2.733	28.883
7705	solar_Tue_Jun_13_9__46__59_2017_L_0.0273312333...	2017-06-13 09:46:59	2.733	28.883
19628	solar_Tue_Jun_13_9__47__4_2017_L_0.03979859380...	2017-06-13 09:47:04	3.980	30.063
4337	solar_Tue_Jun_13_9__47__9_2017_L_0.03979859380...	2017-06-13 09:47:09	3.980	30.063
...	...	...	...	...
23768	solar_Fri_Jun_30_15__53__27_2017_L_0.567408482...	2017-06-30 15:53:27	56.741	24.318
10523	solar_Fri_Jun_30_15__53__32_2017_L_0.567408482...	2017-06-30 15:53:32	56.741	24.318
2043	solar_Fri_Jun_30_15__54__7_2017_L_0.5657411764...	2017-06-30 15:54:07	56.574	29.167
23347	solar_Fri_Jun_30_15__54__12_2017_L_0.565741176...	2017-06-30 15:54:12	56.574	29.167
33432	solar_Fri_Jun_30_15__58__0_2017_L_0.5414267711...	2017-06-30 15:58:00	54.143	31.740

45754 rows × 4 columns

Figura 4: Dataframe

Como comentamos previamente, los autores que crearon el conjunto de datos dieron los valores de pérdida de potencia e irradiancia en el rango [0,1]. Sin embargo, durante la creación del dataframe, hemos convertido esos valores en tanto por cien, para que se representen en el rango [0,100]. También, se ha extraído la información de la fecha y hora de los nombres de las imágenes y se ha representado en formato AAAA-MM-DD HH:MM:SS.

El siguiente paso consiste en el análisis de los datos que vamos a usar en el proyecto, apoyándonos en la información recopilada en el dataframe.

Primeramente, comprobamos el tipo de datos que contiene cada columna:

```
[ ]  df.info()

     <class 'pandas.core.frame.DataFrame'>
     RangeIndex: 45754 entries, 0 to 45753
     Data columns (total 4 columns):
      #   Column      Non-Null Count  Dtype
     ---  ------      --------------  -----
      0   Image Name  45754 non-null  object
      1   Date        45754 non-null  object
      2   Power Loss  45754 non-null  float64
      3   Irradiance  45754 non-null  float64
     dtypes: float64(2), object(2)
     memory usage: 1.4+ MB
```

Figura 5: Tipo de datos

Podemos ver que la columna Image Name y Date contienen datos de tipo object. En Pandas, el tipo de datos object funciona de forma similar a la cadena de datos en Python.

Por otro lado, tanto las columnas Power Loss como Irradiance contienen datos de tipo float64. En Pandas, el tipo de datos float64 se utiliza para representar datos numéricos con decimales.

Además, podemos ver que todas las columnas contienen 45754 datos no nulos, lo que quiere decir que, para cada imagen, tenemos la información de la fecha y hora en la que fue capturada la imagen, el valor de pérdida de potencia y el valor de irradiancia.

Seguidamente, se analizarán las variables irradiance y power loss.

3.1.1.1 Base de datos: irradiance.

Comenzamos el análisis con la variable irradiance. En primer lugar, se han graficado todos los valores de los que disponemos, que son del mes de junio:

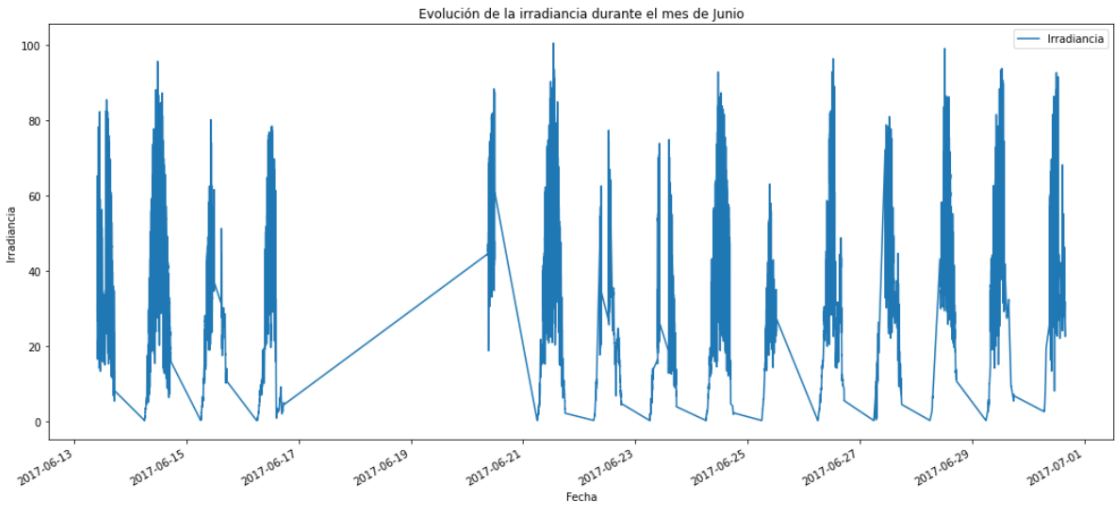


Figura 6: Gráfica irradiancia junio

En un primer vistazo, podemos observar que no disponemos de datos para los días 17, 18, 19 y 20 de junio. Además, se puede observar que la evolución de la irradiancia es similar a lo largo de los días. Para poder observar con más detalle la evolución de la serie temporal, vamos a graficar la irradiancia para varios días distintos:

Para el día 24 de junio:

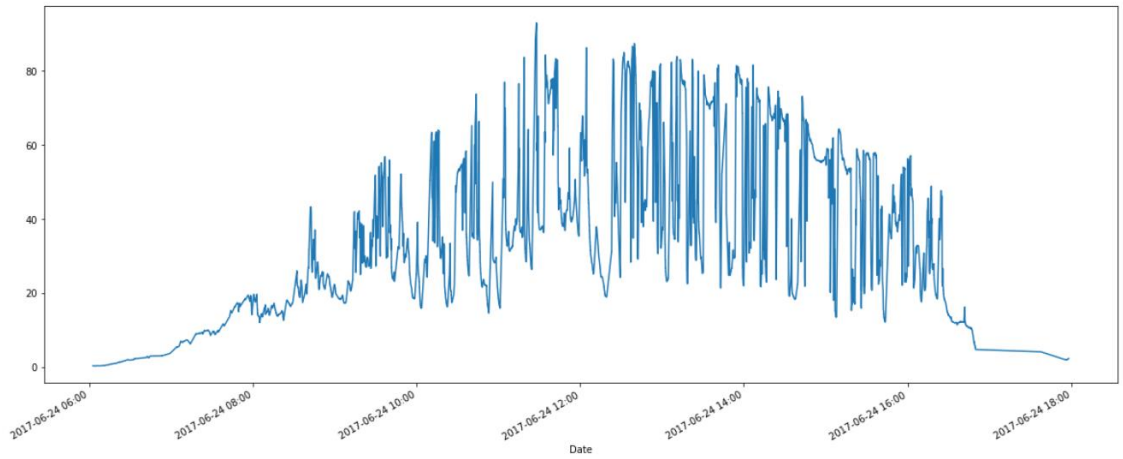


Figura 7: Irradiancia registrada día 24 de junio

Para el día 26:

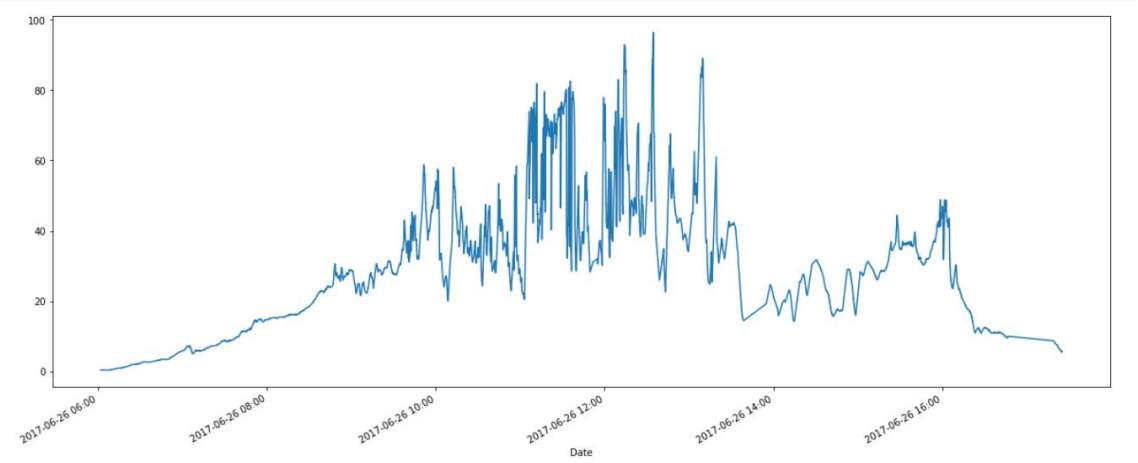


Figura 8: Irradiancia registrada día 26 de junio

La irradiancia se define como la energía por unidad de área de radiación solar incidente en una determinada superficie. Analizando las gráficas, se observa que la irradiancia empieza a tener un valor mayor a medida que avanzan las horas del día y empieza a decrecer a medida que el día se va acabando, lo cual tiene sentido teniendo en cuenta la definición de la magnitud irrandiancia.

A continuación, vamos a buscar datos anómalos en la variable irradiancia. Para ello, vamos a crear un gráfico de cajas:

```
[ ]  # Creamos gráfico de cajas
     boxplot = df.boxplot(column=['Irradiance'])

     # Visualizamos
     boxplot.plot()
```

```
[]
```

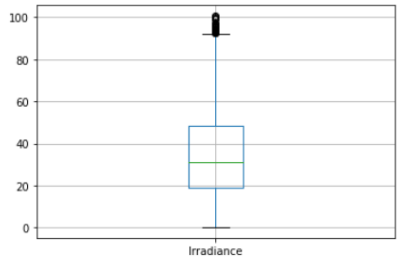


Figura 9: Gráfico de caja irradiancia

Analizando el gráfico de caja, se observa que el límite inferior es 0, mientras que el límite superior aproximadamente está entre 90 y 95. Por definición, cualquier punto que esté más allá del límite inferior o superior sería considerado un outlier. En nuestra gráfica de caja vemos varios valores superiores al límite superior. Sin embargo, los valores de irradiancia vienen dados en el rango [0,100], por lo tanto, solo consideraremos outlier cualquier punto cuyo valor sea superior a 100. Por otra parte, el segmento que divide la caja en dos partes es la mediana. Esto nos permitirá saber si la distribución es simétrica o asimétrica. Como la mediana no se sitúa en el centro de la caja, la distribución no es simétrica. Además, como la parte más larga de la caja es la parte superior a la mediana, se puede afirmar que la distribución tiene asimetría positiva. Esto quiere decir que los datos se concentran en la parte inferior de la distribución, estando más dispersos en la parte superior.

Vamos a buscar los datos anómalos presentes en la variable irradiancia. Recordemos que consideraremos datos anómalos cualquier valor superior a 100:

```
[ ]  df.loc[df["Irradiance"] >= 100]
```

Date	Image Name	Power Loss	Irradiance
2017-06-21 12:59:53	solar_Wed_Jun_21_12__59__53_2017_L_0.039556837...	3.956	100.030
2017-06-21 12:59:59	solar_Wed_Jun_21_12__59__59_2017_L_0.067040325...	6.704	100.613

Figura 10: Datos anómalos irradiancia

Solo hay dos observaciones con valor ligeramente mayor que 100, por lo que no se tomarán medidas correctivas.

3.1.1.2 Base de datos: power loss.

Continuamos el análisis con la variable power loss. En primer lugar, vamos a comprobar si tenemos datos balanceados, es decir, si tenemos un número parecido de muestras para valores bajos, medios y altos de pérdida de potencia. Para ello, se han creado grupos donde agrupar las imágenes cuya pérdida de potencia esté comprendida dentro de un intervalo definido. La agrupación será la siguiente:

- No Damage: Sin pérdida de potencia en el intervalo [0,5].
- Low: Pérdida de potencia baja en el intervalo [5,10].
- Medium: Pérdida de potencia media en el intervalo [10,15].
- High: Pérdida de potencia alta en el intervalo [15,20].
- Extreme: Pérdida de potencia extrema en el intervalo [20,100].

Una vez que tenemos definidos los grupos en los que agrupar las imágenes, se han creado gráficas para poder visualizarlo. Las gráficas son las siguientes:

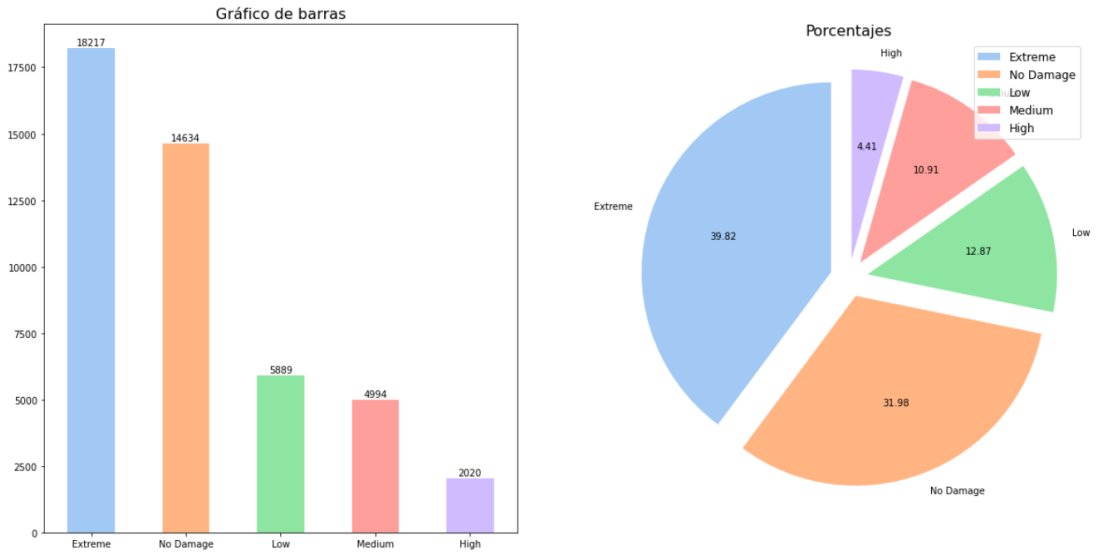


Figura 11: Distribución datos power loss

Observando las gráficas, notamos que los datos no están balanceados, ya que para las categorías Extreme y No Damage, tenemos un mayor número de imágenes si lo comparamos con el número de imágenes que tenemos para las categorías Low, Medium y High.

A continuación, vamos a buscar datos anómalos en la variable power loss. Para ello, vamos a crear un gráfico de cajas:

```
[ ]  # Creamos gráfico de barras
     boxplot = df.boxplot(column=['Power Loss'])

     # Visualizamos
     boxplot.plot()
```

```
[]
```

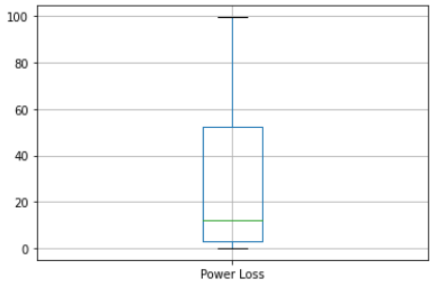


Figura 12: Gráfico de caja power loss

Observando la gráfica anterior puede verse que no hay datos anómalos en nuestros datos de pérdida de potencia. Por otro lado, como se ha comentado previamente, el segmento que divide la caja en dos partes es la mediana. Debido a que la mediana corta la caja en dos lados desiguales, tenemos distribución asimétrica, es decir, falta de simetría en la distribución con respecto de la media. Como la parte más larga de la caja es la parte superior a la mediana, tenemos asimetría positiva o sesgada a la derecha. Esto quiere decir que los datos se concentran en la parte inferior de la distribución, estando más dispersos en la parte superior.

3.2 Generación de datos del modelo de predicción.

Como se ha comentado previamente, el objetivo de nuestro proyecto es desarrollar un modelo usando técnicas de Deep Learning para predecir la pérdida de potencia provocada por la acumulación de suciedad. Para tal fin, vamos a hacer uso del conjunto de datos analizado anteriormente.

3.2.1 Datos de entrada y salida del modelo de predicción.

Como entrada del modelo de predicción, del conjunto de datos usaremos las imágenes de los paneles fotovoltaicos, además de la información de la pérdida de potencia que hemos recogido en el dataframe anterior extrayéndola del nombre de cada imagen. Sin embargo, en este proyecto no vamos a hacer uso ni de la variable Date ni de la variable Irradiance.

Como salida del modelo de predicción, se va a obtener la predicción de la pérdida de potencia del panel fotovoltaico como consecuencia de la acumulación de suciedad en la superficie del panel. La salida será un valor comprendido en el rango [0,100]

3.2.2 División en conjuntos de entrenamiento, validación y prueba.

Como hemos visto en el análisis previo, disponemos de 45754 imágenes en este proyecto. Para construir un modelo fiable, es fundamental disponer de 3 conjuntos de datos diferenciados:

- Entrenamiento: es el conjunto de datos que se utiliza para entrenar y hacer que el modelo aprenda las características o patrones ocultos en los datos. En este proyecto se va a destinar para entrenamiento el 80% del total de imágenes.
- Validación: el conjunto de validación es un conjunto de datos, separado del conjunto de entrenamiento, que se utiliza para validar el rendimiento de nuestro modelo durante el entrenamiento. En este proyecto se va a destinar para entrenamiento el 10% del total de imágenes.
- Prueba: el conjunto de prueba es un conjunto separado de datos que se utiliza para probar el modelo después de completar el entrenamiento. En este proyecto se va a destinar para entrenamiento el 10% del total de imágenes.

Además, a la hora de dividir los datos, es necesario que esto se haga de manera estratificada. Con esta técnica nos aseguramos que tenemos datos balanceados, es decir, que tenemos un número parecido de muestras para valores bajos, medios y altos de pérdida de potencia. Para tal fin, nos ayudamos de la librería split-folders de Python donde indicamos el tanto por ciento de imágenes que queremos en el conjunto de entrenamiento, validación y prueba, en el parámetro ratio:

```
[ ]  splitfolders.ratio('/content/Pictures/Solar_Panel_Soiling_Image_dataset/', output="output", seed=1337, ratio=(0.8, 0.1,0.1))

     Copying files: 45754 files [00:12, 3797.89 files/s]
```

Figura 13: División de datos

En estos momentos, tenemos los conjuntos de datos divididos en entrenamiento, validación y test. Vamos a comprobar que estén balanceados.

Empezamos con el conjunto de entrenamiento:

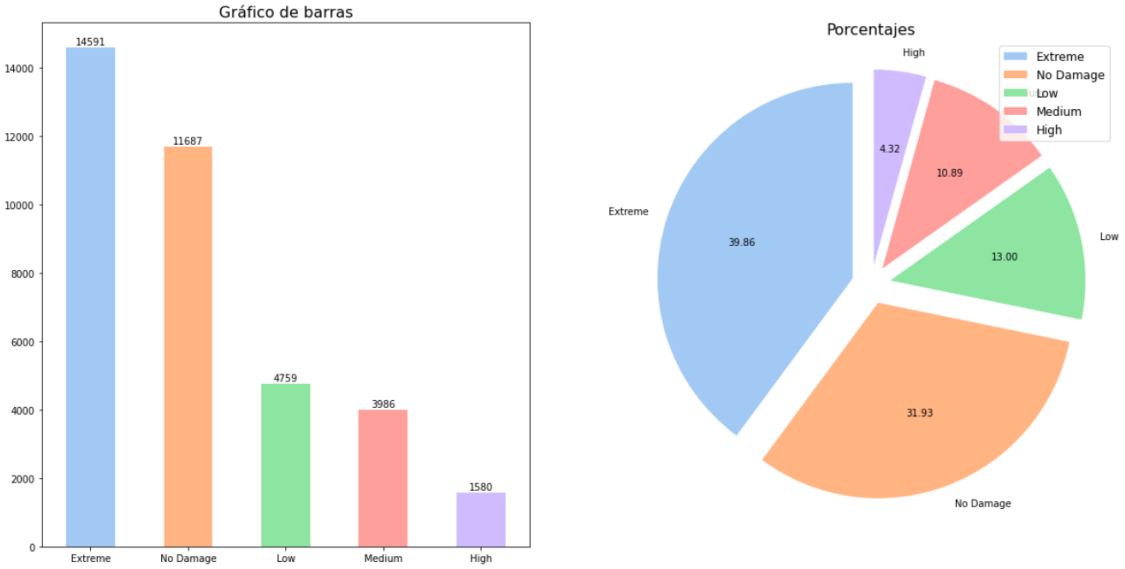


Figura 14: Distribución conjunto de entrenamiento

Seguimos con los datos de validación:

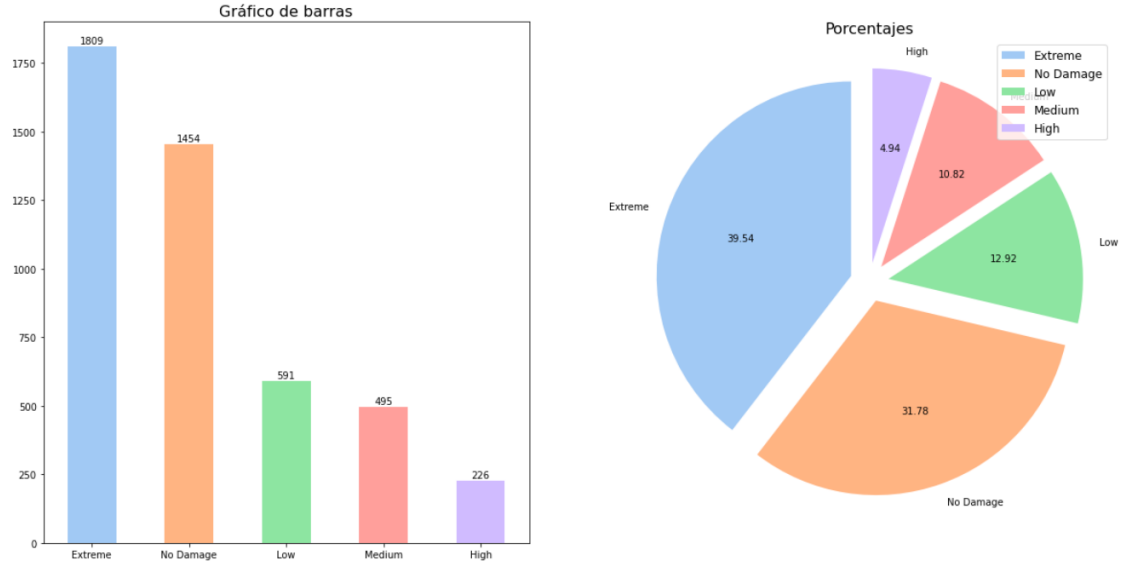


Figura 15: Distribución conjunto de validación

Y acabamos con los datos de test:

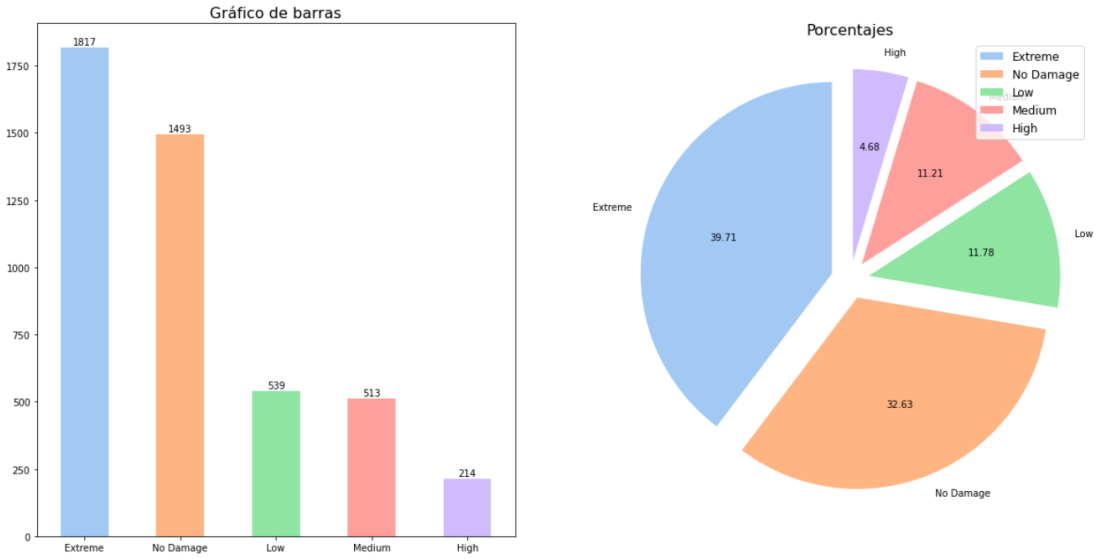


Figura 16: Distribución conjunto de prueba

Observando las gráficas, podemos notar que para entrenamiento, validación y prueba tenemos divididos los datos de manera estratificada, ya que se mantienen las proporciones entre las distintas categorías.

4 Metodología.

En este apartado, se detallará aspectos como la metodología o los modelos de redes neuronales utilizados en el proyecto.

4.1 Propuesta metodológica.

El uso de energías renovables en los últimos años ha revolucionado el sector energético en cualquier parte del mundo. Los objetivos del uso de energías renovables son diversos, pero destacamos los objetivos de la reducción del coste de la energía y la reducción en la emisión de gases contaminantes y de efecto invernadero a la atmósfera.

Una forma de obtención de energía renovable es mediante la transformación de energía solar en energía eléctrica con el uso de paneles fotovoltaicos. Como se ha explicado anteriormente, los paneles fotovoltaicos se basan en el efecto fotoeléctrico, necesitando luz solar para crear una corriente eléctrica en las células fotovoltaicas. Si un panel presenta defectos en la superficie, como acumulación de nieve, excrementos de aves o acumulación de polvo, este efecto fotoeléctrico se verá disminuido, ya que la célula solar recibirá luz solar con menor intensidad. Para limitar esto, es necesario llevar a cabo tareas de mantenimiento de los paneles fotovoltaicos, como puede ser la limpieza de la superficie de los mismos.

Sin embargo, esta tarea puede llegar a ser un problema en zonas alejadas de núcleos urbanos en instalaciones donde se concentran un gran número de placas fotovoltaicas ya que, además de la lejanía, estas instalaciones suelen estar en lugares que no presentan fácil acceso. Por lo tanto, enviar personal para acometer las tareas de limpieza a estas instalaciones puede tener un coste elevado.

Para abordar este problema, en este proyecto se ha planteado la construcción de un sistema que sea capaz de analizar fotografías de paneles fotovoltaicos para predecir la pérdida de potencia provocada por la acumulación de suciedad en la superficie del panel. Con esta información, se podrían programar las tareas de mantenimiento para limitar las pérdidas de rendimiento del panel fotovoltaico provocadas por el problema de la suciedad y, al mismo tiempo, solo enviar personal para ejecutar las tareas de mantenimiento cuando las pérdidas de potencia empiecen a no ser aceptables, con lo que también se conseguiría una reducción en el coste de este servicio.

En la fase de diseño del sistema de predicción, vamos a utilizar dos modelos de redes neuronales. El primer modelo de red, a la que llamaremos CNN propuesta, será una red neuronal convolucional no demasiado compleja que se va a construir con ayuda artículo *Automatic detection of crohn disease in wireless capsule endoscopic images using a deep convolutional neural network* (Diego Marin Santos, Manuel E. Gegundez-Arias, Juan A. Contreras Fernandez, Isaac Perez Borrero,Hector Pallares Manrique, 2022), publicado en:

https://link.springer.com/content/pdf/10.1007/s10489-022-04146-3.pdf

En el artículo se especifica que esta red está diseñada para resolver un problema de clasificación por lo que, para poder usarla en el problema que estamos abordando en este proyecto, será necesario adaptar esta red a un problema de regresión. Para ello, tan solo hace faltar modificar la capa final destinada a la predicción. Esto se detallará más adelante.

El primer modelo de red que vamos a usar es la conocida red VGG16. Esta red la usaremos con la técnica de transfer learning, que consiste en aprovechar una red ya creada y entrenada sobre un problema con características similares a la tarea que abordamos en este proyecto. Como acabamos de comentar, la red ya ha sido entrenada con datos, por lo que necesitaremos cargar los pesos ya ajustados, que en el caso de la red VGG16 serán los pesos *Imagenet,* aunque sin incluir la capa final, ya que necesitamos adaptar la red a nuestro problema de regresión. Esto se hace para recuperar los pesos de las capas de convolución y entrenar las últimas capas que añadimos.

4.2 Modelos de redes utilizados.

En este apartado vamos a analizar la estructura de las redes neuronales convolucionales usadas en el proyecto.

4.2.1 CNN propuesta.

La primera red que vamos a analizar es la red neuronal convolucional construida a partir de los detalles dados en el artículo anterior y modificada para adaptarla a nuestro problema.

La red se ha diseñado para ser alimentada con imágenes RGB de tamaño 192*192. Esta sería la primera modificación con respecto a la red del artículo, ya que necesitamos adaptarla al tamaño de nuestras imágenes.

Siguiendo con la estructura de la red, podemos ver que está formada por 6 bloques con estructura similar, como se muestra en el siguiente esquema de red:

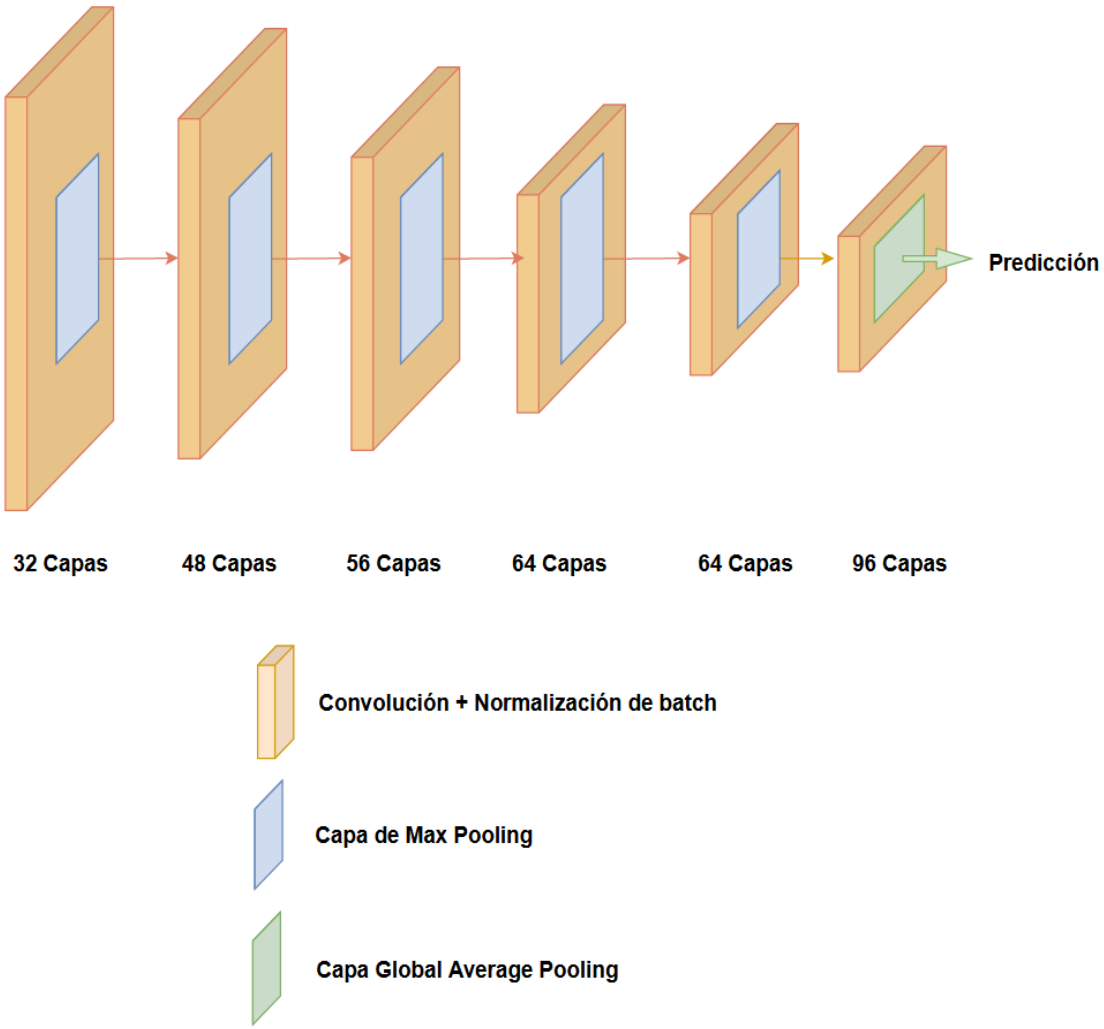


Figura 17: Estructura CNN propuesta

En cada bloque se hacen las siguientes operaciones:

- Convolución: se realiza una operación de convolución al tensor de entrada, con un kernel de tamaño 3x3, stride de tamaño 1 y padding de 1.
- Normalización de Batch: se aplica para acelerar y facilitar la convergencia en el entrenamiento.
- ReLU: se aplica a la salida de la normalización del batch.

La única diferencia entre cada bloque radica en el número de capas de convolución incluidas, que progresivamente aumentan de 32 a 96.

También podemos ver que, al final de los primeros 5 bloques, se incluye una capa de pooling para reducir el tamaño de datos generados. Para ello, se realiza una operación de MaxPooling de tamaño 3x3, stride de 2 y padding de 1.

Por el contrario, al final del último bloque, se aplica una operación de GlobalAveragePooling. Además, en este último bloque, es donde tenemos que adaptar la red para que la predicción sea un valor numérico en lugar de una clase perteneciente a una categoría, es decir, para que se trate de un problema de regresión, en lugar del problema de clasificación que se resuelve en el artículo. Para ello, se añaden dos capas densas, una con 20 neuronas y función de activación relu, y la capa densa final que tendrá una neurona y función de activación relu. Esta última capa está destinada a dar la predicción de pérdida de potencia.

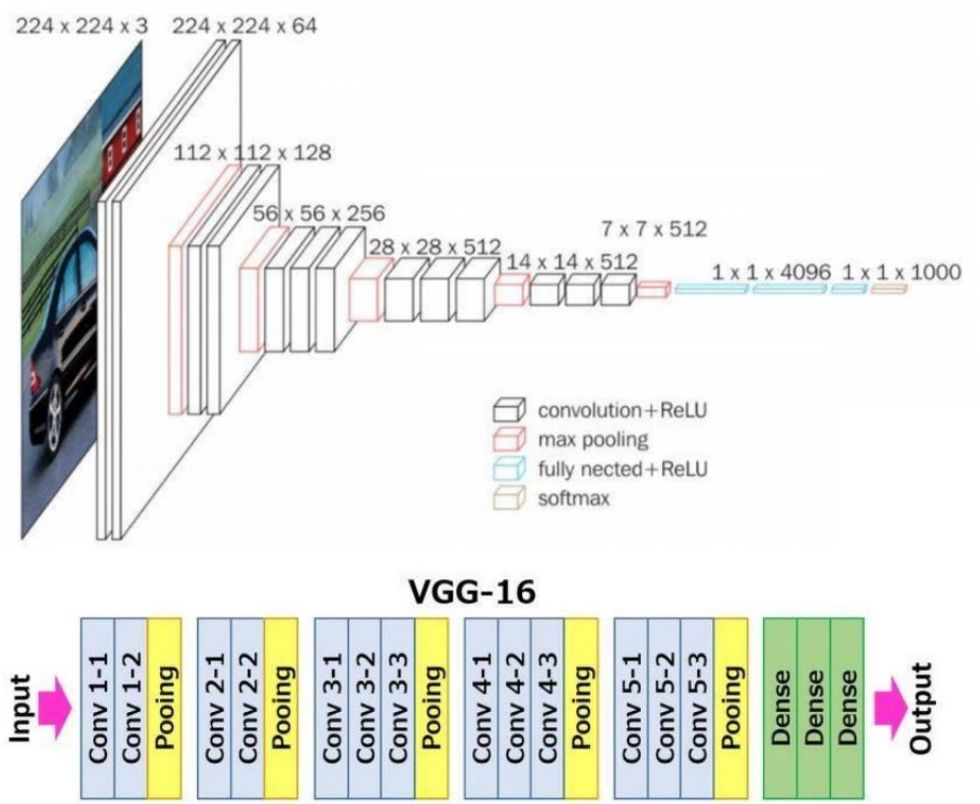
4.2.2 VGG16.

La segunda red que vamos a analizar es la VGG16, que hemos usado en el proyecto apoyándonos con técnicas de transfer learning.

La red VGG16, cuyos autores son Karen Simonyan y Andrew Zisserman, fue presentada en el desafío ImageNet con el objetivo de clasificar imágenes. Esta red es capaz de clasificar 1000 imágenes de 1000 categorías diferentes con un 92,7 % de precisión. Nuestro objetivo es modificar el bloque destinado a la predicción para poder adaptar la red al problema de regresión que estamos tratando de resolver.

La estructura de la red es la siguiente:

Figura 17: Red VGG16



Great Learning. (2022, 5 enero). Everything you need to know about VGG16 - Great Learning. Medium. https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918

Analizando la estructura, podemos ver que esta red necesita imágenes RGB de tamaño 224*224. También, podemos ver que hay trece capas convolucionales, cinco capas de MaxPooling y tres capas densas en el bloque final.

En las capas de convolución, el tamaño del kernel es 3x3, stride de tamaño 1 y padding de 1. La única diferencia entre las capas de convolución la encontramos en el número de filtros, ya que Conv-1 tiene 64 filtros, Conv-2 tiene 128 filtros, Conv-3 tiene 256 filtros y, por último, las capas Conv-4 y Conv-5 tienen 512 filtros.

Además, podemos ver que, al final de la capa de convolución, se aplica una capa de MaxPooling de tamaño 2x2 y stride de 2.

Por último, en el bloque final, la red cuenta con 3 capas densas. Esta última capa es la que tenemos que modificar para adaptarla a nuestro problema de regresión ya que, en estos momentos, la red está preparada para clasificar una imagen entre 1000 categorías distintas.

4.3 Entrenamiento de modelos.

En este apartado, vamos a detallar los pasos dados en el entrenamiento de los modelos.

4.3.1 Aspectos generales.

Primeramente, vamos a definir varios detalles que aplican tanto al entrenamiento de la red CNN propuesta como al entrenamiento de la red VGG16.

Por una parte, comentar que el entrenamiento se ha llevado a cabo de la siguiente manera. El entrenamiento se ha realizado en el entorno de Google Colab. Debido a las limitaciones de uso de una GPU en versiones gratuitas de Colab, no es posible entrenar durante más de 5 horas. Para salvar este impedimento, hemos dividido el entrenamiento en varias fases, de manera que en cada fase hemos usado un número de épocas tal que el tiempo de entrenamiento total no superase las 4 horas. Cuando el modelo había acabado con el entrenamiento en una fase, procedíamos a guardar la red y los valores de pérdidas en un fichero CSV. El objetivo de guardar la red es poder seguir entrenándola en la siguiente fase de entrenamiento con los pesos que se obtuvieron en la fase presente, mientras que el objetivo de guardar los valores de pérdidas en el fichero CSV, es para poder graficar en una única gráfica dichos valores a lo largo del número total de épocas usadas en el entrenamiento.

Por otra parte, destacar que se han llevado a cabo dos entrenamientos, uno en el que no se han usado técnicas de data augmentation y otro en el que sí se ha usado data augmentation. La técnica de data augmentation se utiliza con el objetivo de generar más datos de entrenamiento a partir de los datos de los que disponemos, mediante una serie de transformaciones. El principal motivo para hacer esto es evitar que el modelo se exponga a la misma imagen en diferentes épocas de entrenamiento, lo que puede ayudar a que el modelo aprenda más patrones y pueda generalizar mejor ante nuevos datos.

4.3.2 Preparación para entrenamiento de modelos.

En primer lugar, se va a definir la función de pérdida, métrica de evaluación, optimizador, hiperparámetros o métodos de callbacks usados durante el entrenamiento. Toda la configuración que se detalla en este apartado, se ha usado tanto para la red CNN propuesta como para la red VGG16.

Empezamos definiendo la función de pérdida. La función de pérdida evalúa la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas durante el aprendizaje para optimizar la función de aprendizaje. Mientras más bajo sea este valor, más eficiente será la red neuronal. En nuestro proyecto, se ha seleccionado la métrica error absoluto medio, o mean absolute error en inglés, ya que estamos tratando con un problema de regresión. Esta métrica calcula la diferencia media entre los valores calculados y los valores reales.

A continuación, pasamos a definir el optimizador. El optimizador es un método usado para actualizar los parámetros de la red con el objetivo de optimizarlos y reducir las pérdidas o errores cometidos por la red. En nuestro proyecto, se ha seleccionado el optimizador Adam.

Con respecto a la métrica de evaluación, se trata de una métrica usada al final del entrenamiento para medir el rendimiento global del modelo. En este proyecto, se ha elegido la métrica error absoluto medio.

Seguimos definiendo el hiperparámetro batch size usado en el proyecto. Este hiperparámetro define el número de muestras que se propagarán a través de la red. En nuestro caso, hemos establecido este hiperparámetro con el valor 32.

Por último, definimos los llamados métodos de callbacks. Se trata de un método de Keras que puede ejecutar acciones durante la fase de entrenamiento. Vamos a definir varias acciones en este proyecto.

En primer lugar, definimos una función que nos guardará el modelo que menor valor de pérdida consiga durante el entrenamiento. Para ello, usaremos la librería ModelCheckPoint de Keras. Con el uso del parámetro filepath, podemos establecer como queremos que se nombre el modelo guardado. Para una mayor claridad, haremos que el modelo se guarde con la época y el valor de pérdidas, como puede verse en la imagen:

```
# Establecemos el nombre del modelo guardado
filepath = 'my_best_model.epoch{epoch:02d}-loss{val_loss:.2f}.hdf5'

# Creamos una instancia de ModelCheckPoint, eligiendo que solo se guarde el mejor modelo
best_model = ModelCheckpoint(filepath=filepath,
                             monitor='val_loss',
                             verbose=1,
                             save_best_only=True,
                             mode='min')
```

Figura 18: ModelCheckPoint

Además, vamos a definir una función que pare el entrenamiento si el modelo no ha mejorado el valor de pérdidas a lo largo de un determinado número de épocas, con el uso del parámetro patience. Para ello, usamos el método EarlyStopping:

```
earlystop = EarlyStopping(patience=50)
```

Figura 19: EarlyStopping

También, vamos a definir una función que reduzca el valor de learning rate en un factor de 0.5 si el modelo no ha mejorado el valor de pérdidas a lo largo un determinado número de épocas, con el uso del parámetro patience. Para ello, usamos el método ReduceLROnPlateau:

```
learning_rate_reduction = ReduceLROnPlateau(monitor='mean_absolute_error',
                                            patience=10,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.0001)
```

Figura 20: ReduceLROnPlateau

Estas 3 funciones componen el método de callback que se usará durante el entrenamiento. Lo guardamos en la variable callbacks:

```
callbacks = [earlystop, best_model, learning_rate_reduction]
```

Figura 21: Callbacks

4.3.3 Entrenamiento sin data augmentation.

En este apartado, se va a detallar el proceso de entrenamiento tanto de la red CNN propuesta como de la red VGG16 sin usar técnicas de data augmentation por lo que, en esta ocasión, las imágenes no van a ser transformadas.

Comenzaremos detallando el proceso de generación de imágenes, que será idéntico para ambas redes neuronales.

Para alimentar a las redes en el entrenamiento, vamos a usar un generador de imágenes. Esta técnica permite preprocesar los datos de forma personalizada para cada bucle de entrenamiento, como por ejemplo usando técnicas de aumento de datos, aunque la técnica de aumento de datos se usará posteriormente. Otra ventaja de usar un generador de imágenes es que permite usar gradualmente el conjunto de datos de entrenamiento si su tamaño no permite cargar todo en la RAM.

Definimos un generador de imágenes para el conjunto de datos de entrenamiento, otro para el conjunto de validación y otro para el conjunto de prueba. Para ello, usamos la función de keras ImageDataGenerator. Como hemos comentado previamente, las técnicas de aumento de datos no se usarán de momento, por lo que solo vamos a reescalar los datos. El objetivo de reescalar los datos radica en que nuestras imágenes originales consisten en coeficientes RGB en el rango [0,255], pero tales valores son demasiado altos para que nuestro modelo los procese, por lo que vamos a reescalar para que los valores de los coeficientes RGB estén en el rango [0,1].

Esto puede verse en la siguiente imagen:

```
# Generador datos entrenamiento
train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
)

# Generador datos validación
val_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
)

# Generador datos test
test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
)
```

Figura 22. Image Data Generator CNN sin data augmentation

Además, es necesario definir el directorio donde están las imágenes. Para el caso de las imágenes destinadas a entrenamiento, el directorio será */content/output/train/PanelImages/*. Para las imágenes destinadas a validación, el directorio será */content/output/val/PanelImages/*, mientras que, para las imágenes destinadas a test, el directorio será */content/output/test/PanelImages/*.

Por otro lado, como usaremos el generador para alimentar a la red, nos ayudaremos del dataframe creado para especificar el valor de la variable X, que será el nombre de la imagen dentro del directorio, y el valor Y que se pretende predecir, que será la pérdida de potencia de esa imagen en concreto.

Esto puede verse en la siguiente imagen:

```
# Alimentamos el generador de imágenes con datos de entrenamiento
train_images = train_generator.flow_from_dataframe(
    dataframe=df_train_estratificado,
    directory='/content/output/train/PanelImages/',
    x_col='Image Name',
    y_col='Power Loss',
    target_size=(192, 192),
    color_mode='rgb',
    class_mode='raw',
    batch_size=32,
    shuffle=False,
    seed=42
)

# Alimentamos el generador de imágenes con datos de validación
val_images = val_generator.flow_from_dataframe(
    dataframe=df_val_estratificado,
    directory='/content/output/val/PanelImages/',
    x_col='Image Name',
    y_col='Power Loss',
    target_size=(192, 192),
    color_mode='rgb',
    class_mode='raw',
    batch_size=32,
    shuffle=False,
    seed=42
)
# Alimentamos el generador de imágenes con datos de test
test_images = test_generator.flow_from_dataframe(
    dataframe=df_test_estratificado,
    directory='/content/output/test/PanelImages/',
    x_col='Image Name',
    y_col='Power Loss',
    class_mode="raw",
    shuffle=False,
    target_size=(192, 192),  # size of the image
    batch_size=1,  # use only one image for visualization
)
```

Figura 23: Generación de imágenes CNN sin data augmentation

Además, graficamos varias imágenes de paneles fotovoltaicos sin data augmentation, junto con los valores de pérdida de potencia:

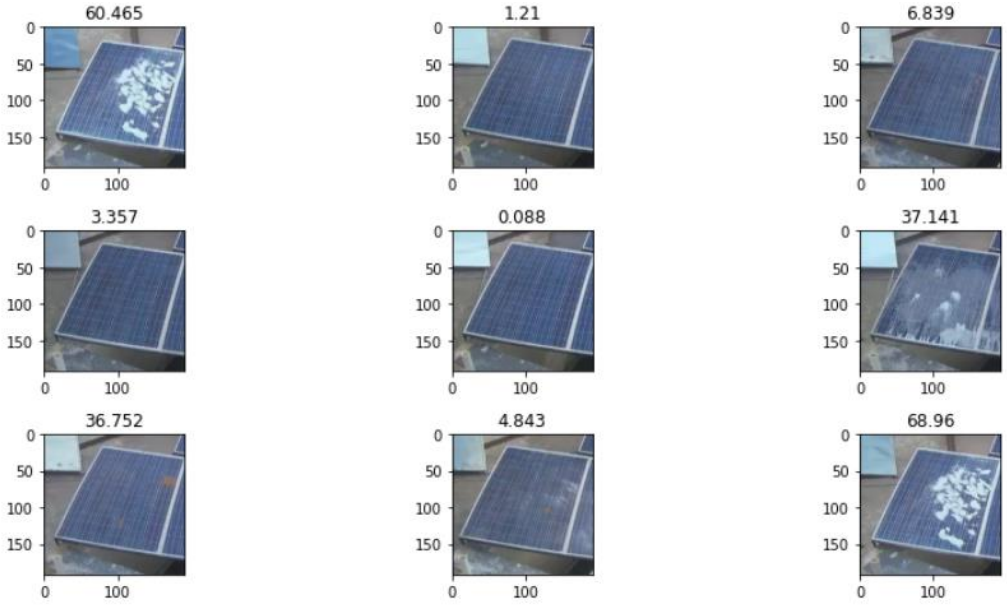


Figura 24: Imágenes sin data augmentation

4.3.3.1 Entrenamiento CNN propuesta.

Ya estamos en disposición de empezar el entrenamiento de nuestro modelo, cuya estructura es:

```
model = tf.keras.Sequential(
    [

    # BLOQUE 1

    tf.keras.layers.Conv2D(input_shape=(192, 192, 3), kernel_size=(3, 3), padding='same', strides = 1, filters = 32),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.ReLU(),
    tf.keras.layers.MaxPooling2D(pool_size = (3, 3), padding='same', strides = 2),

    # BLOQUE 2

    tf.keras.layers.Conv2D(kernel_size=(3, 3), padding='same', strides = 1, filters = 48),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.ReLU(),
    tf.keras.layers.MaxPooling2D(pool_size = (3, 3), padding='same', strides = 2),

    # BLOQUE 3

    tf.keras.layers.Conv2D(kernel_size=(3, 3), padding='same', strides = 1, filters = 56),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.ReLU(),
    tf.keras.layers.MaxPooling2D(pool_size = (3, 3), padding='same', strides = 2),

    # BLOQUE 4

    tf.keras.layers.Conv2D(kernel_size=(3, 3), padding='same', strides = 1, filters = 64),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.ReLU(),
    tf.keras.layers.MaxPooling2D(pool_size = (3, 3), padding='same', strides = 2),


    # BLOQUE 5

    tf.keras.layers.Conv2D(kernel_size=(3, 3), padding='same', strides = 1, filters = 64),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.ReLU(),
    tf.keras.layers.MaxPooling2D(pool_size = (3, 3), padding='same', strides = 2),

    # BLOQUE 6

    tf.keras.layers.Conv2D(kernel_size=(3, 3), padding='same', strides = 1, filters = 96),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.ReLU(),
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(20,activation='relu'),
    tf.keras.layers.Dense(1,activation='relu')
    ]
)
```

Figura 25: Modelo CNN propuesto

Recordemos que, debido a las limitaciones de uso de una GPU en Google Colab, el entrenamiento se ha dividido en varias fases. En cada una de las fases, el modelo se ha entrenado durante 4 horas aproximadamente lo que, para la CNN propuesta sin data augmentation, equivale a 80 épocas.

Para entrenar el modelo, usaremos el método .fit() de Keras. Especificamos los datos de entrenamiento, que serán alimentados con el generador de imágenes de entrenamiento, el tamaño del batch, el número de épocas, elegimos que los datos se barajen, le indicamos que se debe usar el método de callback previamente definido y, por último, especificamos las imágenes que se usarán para ir validando cómo de bien se comporta el modelo con datos nuevos que no han sido usados en el entrenamiento. Los resultados obtenidos durante el entrenamiento se guardarán en la variable history.

```
history = model.fit(train_images, batch_size = BATCH_SIZE ,epochs = 80, shuffle=True, callbacks = callbacks, validation_data=val_images)
```

Figura 26: Entrenamiento CNN sin data augmentation

Este proceso se ha repetido durante 3 fases, guardando en cada fase los resultados de los pesos ajustados de la red y los resultados de la variable history en un fichero CSV. La variable history contiene los valores de pérdidas en entrenamiento y validación durante cada fase de entrenamiento, lo cual será necesario para poder crear la gráfica con el resultado global obtenido durante las fases que ha durado el entrenamiento.

4.3.3.2 Entrenamiento VGG16.

Para entrenar el modelo VGG16, primero hay que modificar la última capa de la red para adaptarla a nuestro problema. Empezamos importando el modelo y cargamos los pesos Imagenet. Además, especificamos el tamaño de imagen con la que se va a alimentar la red e indicamos que no se incluya la capa final usada para hacer predicciones. Esto es necesario para adaptar la red a un problema de regresión y, también, para adaptar los pesos de la capa final a nuestros datos:

```
base_model = VGG16(weights="imagenet", include_top=False, input_shape=(192, 192, 3))
```

Figura 27: Modelo VGG16 modificado

Seguidamente, congelamos todos los pesos para que no se modifiquen durante el entrenamiento, ya que solo queremos que se modifiquen los pesos de la capa final:

```
base_model.trainable = False
```

Figura 28: Congelación pesos red

Construimos el bloque final, añadiendo una capa densa de 20 neuronas con función de activación relu, y una capa final densa de 1 neurona con función de activación relu, usada para hacer la predicción:

```
flatten_layer = layers.Flatten()
dense_layer_1 = layers.Dense(20, activation='relu')
prediction_layer = layers.Dense(1, activation='relu')
```

Figura 29: VGG16 capa final predicción

Finalmente, creamos el modelo:

```
model = models.Sequential([
    base_model,
    flatten_layer,
    dense_layer_1,
    prediction_layer
])
```

Figura 30: Modelo VGG16 adaptado

Con esto, ya podemos entrenar el modelo. Recordemos que, debido a las limitaciones de uso de una GPU en Google Colab, el entrenamiento se ha dividido en varias fases. En cada una de las fases, el modelo se ha entrenado durante 4 horas aproximadamente lo que, para la CNN propuesta sin data augmentation, equivale a 80 épocas.

Para entrenar el modelo, usaremos el método .fit() de Keras, de igual manera que se hizo en el caso de la red CNN:

```
history = model.fit(train_images, batch_size = BATCH_SIZE ,epochs = 80, shuffle=True, callbacks = callbacks, validation_data=val_images)
```

Figura 31: Entrenamiento VGG16 sin data augmentation

Este proceso se ha repetido durante 3 fases, guardando en cada fase los resultados de los pesos ajustados de la red y los resultados de la variable history en un fichero CSV.

4.3.4 Entrenamiento con data augmentation.

En este apartado, se va a detallar el proceso de entrenamiento tanto de la red CNN propuesta como de la red VGG16 usando técnicas de data augmentation por lo que, en esta ocasión, las imágenes van a ser transformadas.

Comenzaremos detallando el proceso de generación de imágenes, que será idéntico para ambas redes neuronales.

De la misma manera que para el caso analizado anteriormente, para alimentar a la red en el entrenamiento, vamos a usar un generador de imágenes para el conjunto de datos de entrenamiento, otro generador de imágenes para el conjunto de datos de validación y otro generador de imágenes para el conjunto de prueba. Para ellos, usamos la función de keras ImageDataGenerator.

En primer lugar, vamos a reescalar los datos. El objetivo de reescalar los datos radica en que nuestras imágenes originales consisten en coeficientes RGB en el rango [0,255], pero tales valores son demasiado altos para que nuestro modelo los procesara, por lo vamos a reescalar para que los valores de los coeficientes RGB estén en el rango [0,1]. Como técnicas de aumento de datos, vamos a aplicar un zoom del 0.1%, vamos a rotar las imágenes 20 grados, y vamos a voltearlas horizontalmente. Esto solo lo vamos a aplicar al generador de imágenes destinadas al entrenamiento de la red.

Esto puede verse en la siguiente imagen:

```
# Generador datos entrenamiento
train_generator_aug = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255,
    zoom_range=0.1,
    rotation_range = 20,
    horizontal_flip=True,
)

# Generador datos validación
val_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
)

# Generador datos test
test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
)
```

Figura 27: Image Data Generator CNN con data augmentation

Además, es necesario definir el directorio donde están las imágenes. Para el caso de las imágenes destinadas a entrenamiento, el directorio será */content/output/train/PanelImages/*. Para las imágenes destinadas a validación, el directorio será */content/output/val/PanelImages/*, mientras que, para las imágenes destinadas a test, el directorio será */content/output/test/PanelImages/*. Por otro lado, como usaremos el generador para alimentar a la red, nos ayudaremos del dataframe creado para especificar el valor de la variable X, que será el nombre de la imagen dentro del directorio, y el valor Y que se pretende predecir, que será la pérdida de potencia de esa imagen en concreto.

Esto puede verse en la siguiente imagen:

```
# Alimentamos el generador de imágenes con datos de entrenamiento
train_images = train_generator_aug.flow_from_dataframe(
    dataframe=df_train_estratificado,
    directory='/content/output/train/PanelImages/',
    x_col='Image Name',
    y_col='Power Loss',
    target_size=(192, 192),
    color_mode='rgb',
    class_mode='raw',
    batch_size=32,
    shuffle=False,
    seed=42
)

# Alimentamos el generador de imágenes con datos de validación
val_images = val_generator.flow_from_dataframe(
    dataframe=df_val_estratificado,
    directory='/content/output/val/PanelImages/',
    x_col='Image Name',
    y_col='Power Loss',
    target_size=(192, 192),
    color_mode='rgb',
    class_mode='raw',
    batch_size=32,
    shuffle=False,
    seed=42
)
# Alimentamos el generador de imágenes con datos de test
test_images = test_generator.flow_from_dataframe(
    dataframe=df_test_estratificado,
    directory='/content/output/test/PanelImages/',
    x_col='Image Name',
    y_col='Power Loss',
    class_mode="raw",
    shuffle=False,
    target_size=(192, 192),  # size of the image
    batch_size=1,  # use only one image for visualization
)
```

Figura 27: Generación de imágenes CNN con data augmentation

Además, graficamos varias imágenes de paneles fotovoltaicos con data augmentation, junto con los valores de pérdida de potencia:

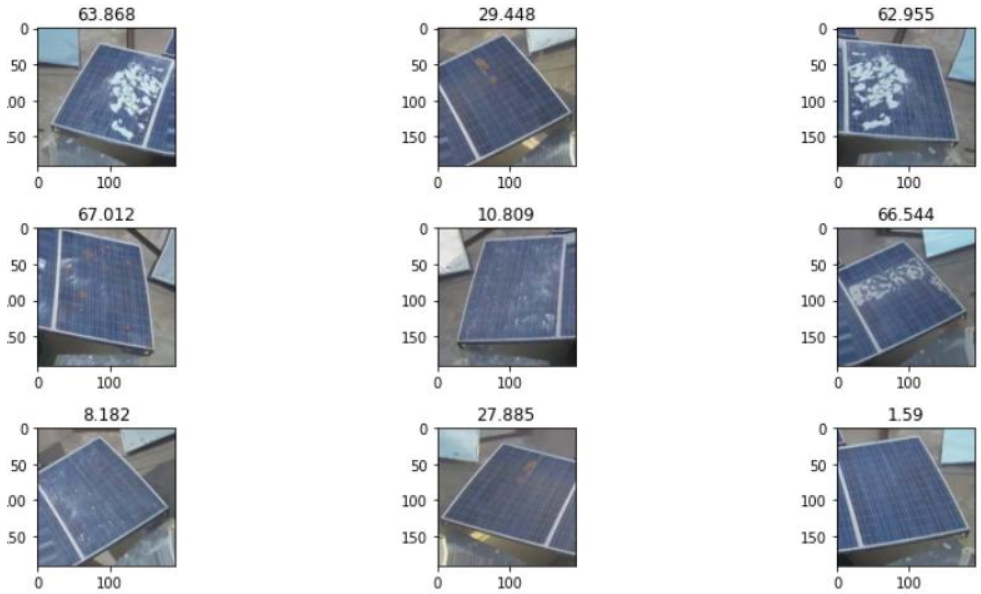


Figura 28: Imágenes con data augmentation

4.3.4.1 Entrenamiento CNN propuesta.

El procedimiento será el mismo que el explicado para el caso del entrenamiento sin usar técnicas de data augmentation.

Solo hay que tener en cuenta una diferencia, que es el número de épocas en las que tenemos que entrenar el modelo. Debido al uso de data augmentation, el tiempo de entrenamiento de una época será mayor por lo que, para no pasar el límite de 4 horas en el uso de una GPU de forma continuada, se ha establecido que cada fase de entrenamiento sea de 40 épocas:

Ya podemos empezar a entrenar el modelo. Para ello, usaremos el método .fit() de Keras. Especificamos los datos de entrenamiento, que serán alimentados con el generador de imágenes de entrenamiento, el tamaño del batch, el número de épocas, elegimos que los datos se barajen, le indicamos que se debe usar el método de callback previamente definido y, por último, especificamos las imágenes que se usarán para ir validando como de bien se comporta el modelo con datos nuevos que no han sido usados en el entrenamiento. Los resultados obtenidos durante el entrenamiento se guardarán en la variable history_aug.

```
history_aug = model.fit(train_images, batch_size = BATCH_SIZE ,epochs = 40, shuffle=True, callbacks = callbacks, validation_data=val_images)
```

Figura 29: Entrenamiento CNN con data augmentation

Este proceso se ha repetido durante 4 fases, guardando en cada fase los resultados de los pesos ajustados de la red y los resultados de la variable history_aug en un fichero CSV.

4.3.4.2 Entrenamiento VGG16.

El procedimiento será el mismo que el explicado para el caso del entrenamiento sin usar técnicas de data augmentation, importando la red VGG16 y adaptándola a nuestro problema.

Solo hay que tener en cuenta una diferencia, que es el número de épocas en las que tenemos que entrenar el modelo. Debido al uso de data augmentation, el tiempo de entrenamiento de una época será mayor por lo que, para no pasar el límite de 4 horas en el uso de una GPU de forma continuada, se ha establecido que cada fase de entrenamiento sea de 40 épocas:

Ya podemos empezar a entrenar el modelo. Para ello, usaremos el método .fit() de Keras. Especificamos los datos de entrenamiento, que serán alimentados con el generador de imágenes de entrenamiento, el tamaño del batch, el número de épocas, elegimos que los datos se barajen, le indicamos que se debe usar el método de callback previamente definido y, por último, especificamos las imágenes que se usarán para ir validando cómo de bien se comporta el modelo con datos nuevos que no han sido usados en el entrenamiento. Los resultados obtenidos durante el entrenamiento se guardarán en la variable history_aug.

```
history_aug = model.fit(train_images, batch_size = BATCH_SIZE ,epochs = 40, shuffle=True, callbacks = callbacks, validation_data=val_images)
```

Figura 30: Entrenamiento VGG16 con data augmentation

5 Resultados.

En este apartado se detallarán los resultados obtenidos para la red CNN propuesta y la red VGG16, tanto para el caso en el que no se ha usado data augmentation como para el caso en el que sí se ha usado data augmentation.

5.1 Resultados CNN propuesta durante el entrenamiento.

Como se ha comentado previamente, debido a limitaciones en el uso de Google Colab, el entrenamiento de los modelos se ha llevado a cabo en varias fases, guardando los resultados de cada fase para poder graficarlos juntos. Por lo tanto, mostraremos y analizaremos la gráfica con los resultados globales.

5.1.1 CNN propuesta sin data augmentation.

Vamos a analizar el entrenamiento llevado a cabo para la red CNN propuesta sin usar técnicas de data augmentation. A continuación, mostramos la gráfica que recoge los valores de pérdida en entrenamiento y validación durante todas las fases del entrenamiento:

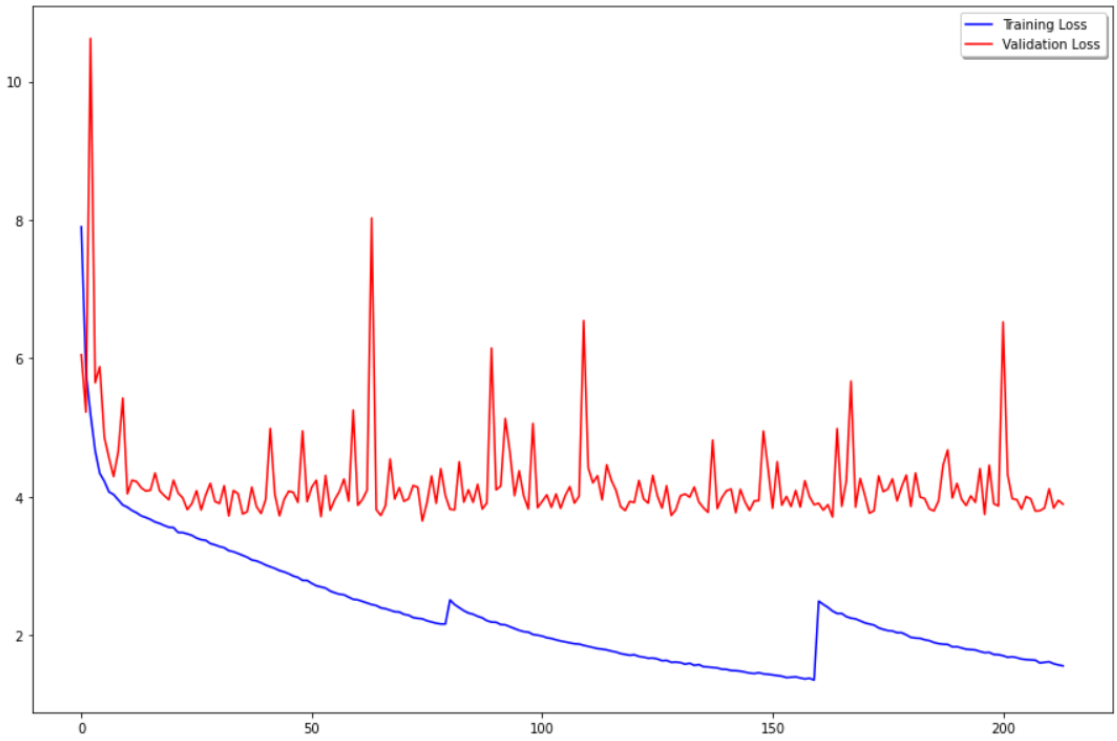


Figura 31: Pérdidas en entrenamiento y validación CNN propuesta sin data augmentation

Observando la curva de pérdida durante la etapa de validación, vemos que durante las distintas épocas los valores de pérdida mayoritariamente oscilan entre 3.7 y 4.5, aproximadamente. Esto nos hace pensar que el modelo no va a mejorar en validación, aunque sigamos entrenando durante mas épocas. Por otra parte, vemos que los valores de pérdida durante la etapa de entrenamiento van disminuyendo durante el paso de las épocas a pesar de que en validación no parece mejorar significativamente, por lo que parece que el modelo está sobreaprendiendo los datos de entrenamiento.

5.1.2 CNN propuesta con data augmentation.

A continuación, vamos a analizar el entrenamiento llevado a cabo para la red CNN propuesta usando técnicas de data augmentation. Seguidamente, mostramos la gráfica que recoge los valores de pérdida en entrenamiento y validación durante todas las fases del entrenamiento:

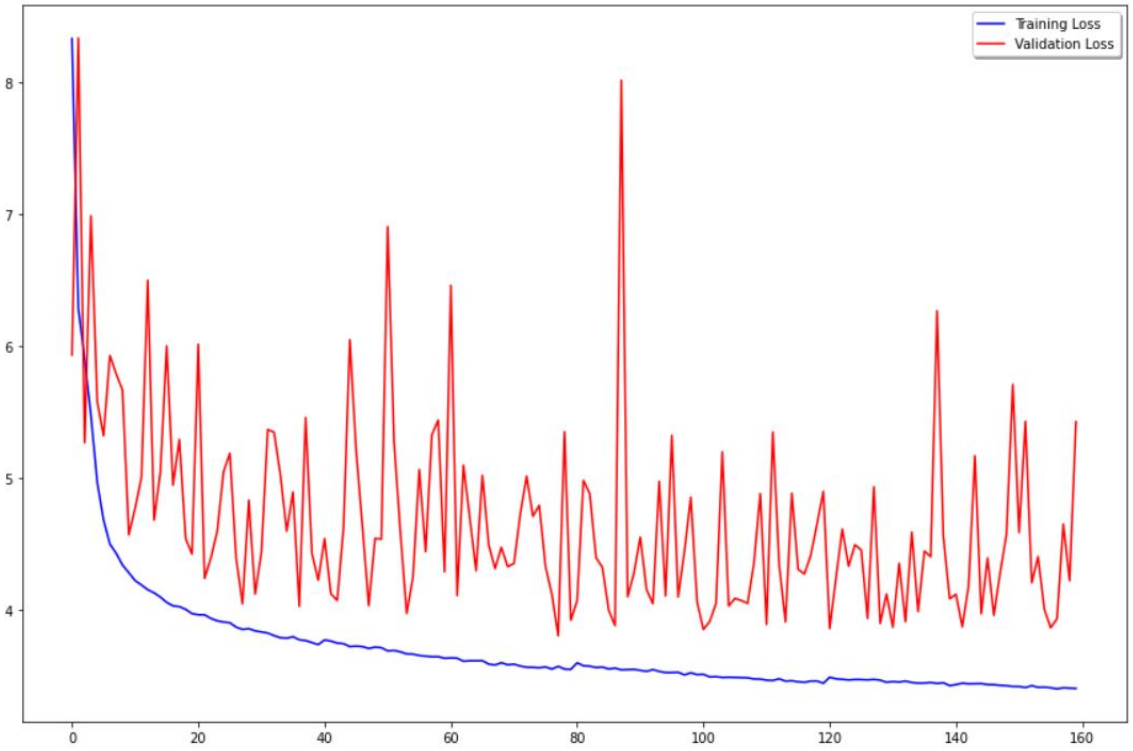


Figura 31: Pérdidas en entrenamiento y validación CNN propuesta con data augmentation

Observando la curva de pérdida durante la etapa de validación, vemos que durante las distintas épocas los valores de pérdida mayoritariamente oscilan entre 4 y 5.5, aproximadamente. Esto nos hace pensar que el modelo no va a mejorar en validación, aunque sigamos entrenando durante mas épocas. Además, podemos observar variaciones grandes en los valores de pérdida entre una época y la siguiente. Parece que el modelo no es capaz de generalizar ante nuevos datos.

Por otra parte, vemos que los valores de pérdida durante la etapa de entrenamiento van disminuyendo durante el paso de las épocas a pesar de que en validación no parece mejorar significativamente, por lo que se deduce que el modelo está sobreaprendiendo los datos de entrenamiento.

5.2 Resultados VGG16 durante el entrenamiento.

Al igual que para la red CNN propuesta, el entrenamiento de los modelos se ha llevado a cabo en varias fases, guardando los resultados de cada fase para poder graficarlos juntos. Por lo tanto, mostraremos y analizaremos la gráfica con los resultados globales.

5.2.1 VGG16 sin data augmentation.

Vamos a analizar el entrenamiento llevado a cabo para la red VGG16 sin usar técnicas de data augmentation. A continuación, mostramos la gráfica que recoge los valores de pérdida en entrenamiento y validación durante todas las fases del entrenamiento:

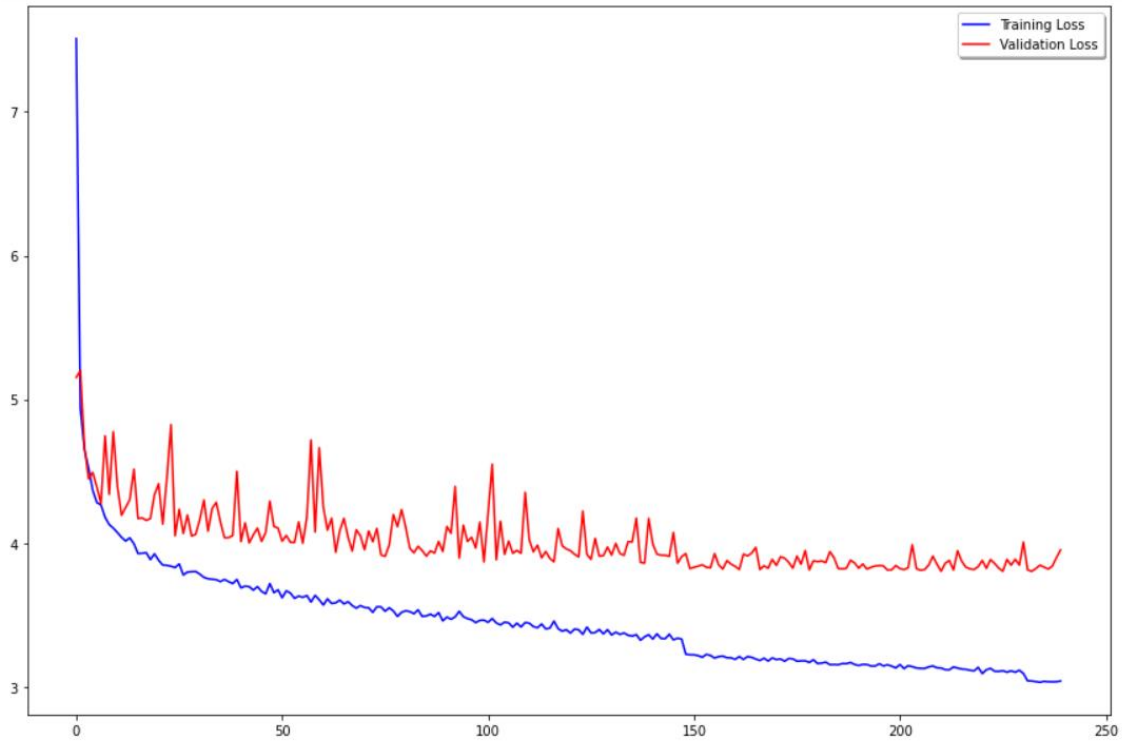


Figura 32: Pérdidas en entrenamiento y validación VGG16 sin data augmentation

Observando la curva de pérdida durante la etapa de entrenamiento y validación, vemos que, a lo largo de las distintas épocas, los valores de pérdida van disminuyendo progresivamente. Al final de la etapa de validación, vemos que los valores de pérdida se sitúan en torno a 3.8, mientras que los valores de pérdida en el entrenamiento acaban en torno a 3.1. También, observamos que los valores de pérdida en validación y entrenamiento parecen ir estabilizándose con el paso de las épocas, sin mostrar grandes oscilaciones entre una época y la siguiente, como pudo apreciarse en el caso de la CNN propuesta.

5.2.2 VGG16 con data augmentation.

A continuación, vamos a analizar el entrenamiento llevado a cabo para la red VGG16 usando técnicas de data augmentation. Seguidamente, mostramos la gráfica que recoge los valores de pérdida en entrenamiento y validación durante todas las fases del entrenamiento:

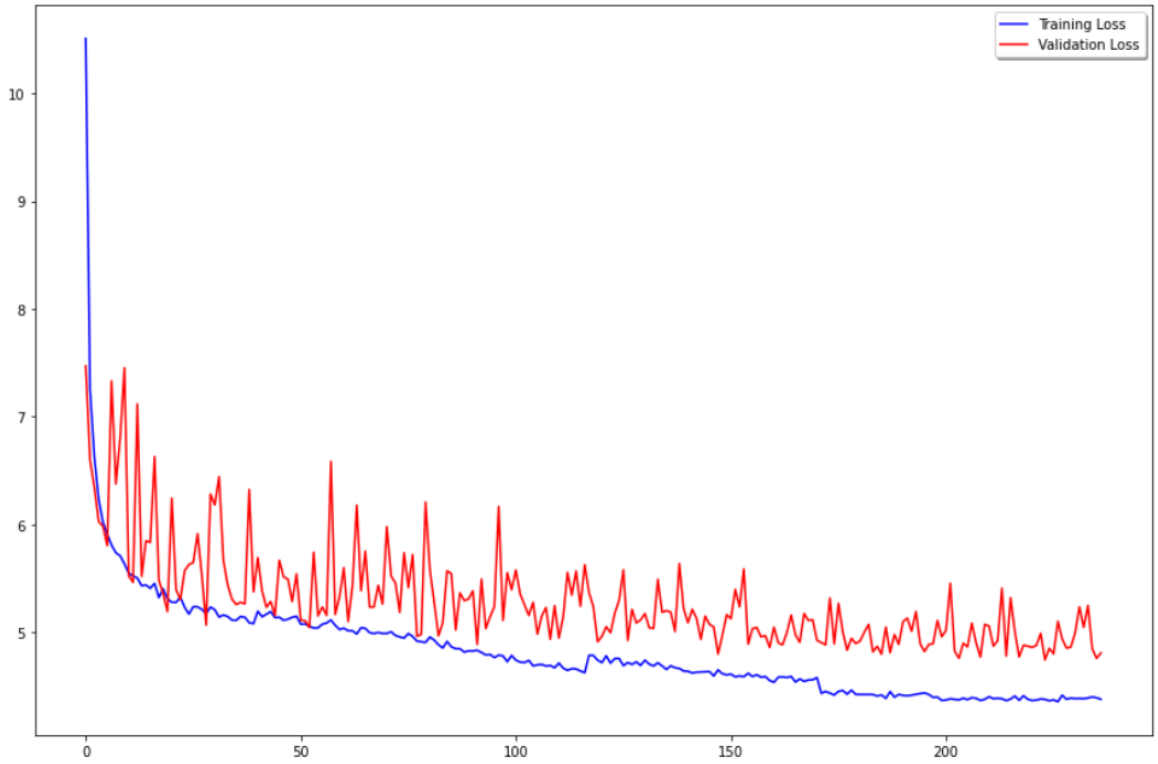


Figura 33: Figura 32: Pérdidas en entrenamiento y validación VGG16 con data augmentation

Observando la curva de pérdida durante la etapa de entrenamiento y validación, vemos que, a lo largo de las distintas épocas, los valores de pérdida van disminuyendo progresivamente. Al final de la etapa de validación, vemos que los valores de pérdida se sitúan en torno a 4.8, mientras que los valores de pérdida en entrenamiento acaban en torno a 4.3. También, observamos que los valores de pérdida en validación y entrenamiento parecen ir estabilizándose con el paso de las épocas, aunque esta estabilización es más apreciable en la curva de entrenamiento que en la de validación.

5.3 Resultados CNN propuesta en el conjunto de prueba.

Seguidamente, vamos a detallar los resultados obtenidos sobre el conjunto de datos de prueba de la red CNN propuesta.

5.3.1 CNN propuesta sin data augmentation.

El resultado obtenido para la red CNN propuesta entrenada sin usar técnicas de data augmentation ha sido:

```
[ ]  model.evaluate(test_images)

     4576/4576 [==============================] - 33s 5ms/step - loss: 3.7298 - mean_absolute_error: 3.7298
     [3.729844808578491, 3.729844808578491]
```

Figura 33: Evaluación CNN propuesta sin data augmentation

La métrica usada para evaluar la red ha sido el error absoluto medio. Vemos que el valor obtenido es 3.729.

5.3.2 CNN propuesta con data augmentation.

El resultado obtenido para la red CNN propuesta entrenada usando técnicas de data augmentation ha sido:

```
[ ]  model.evaluate(test_images)

     4576/4576 [==============================] - 28s 5ms/step - loss: 3.8280 - mean_absolute_error: 3.8280
     [3.827977418899536, 3.827977418899536]
```

Figura 34: Evaluación CNN propuesta con data augmentation

La métrica usada para evaluar la red ha sido el error absoluto medio. Vemos que el valor obtenido es 3.827.

5.4 Resultados VGG16 en el conjunto de pruebas.

Seguidamente, vamos a detallar los resultados obtenidos sobre el conjunto de datos de prueba de la red VGG16.

5.4.1 VGG16 sin data augmentation.

El resultado obtenido para la red VGG16 entrenada sin usar técnicas de data augmentation ha sido:

```
model.evaluate(test_images)

4576/4576 [==============================] - 49s 9ms/step - loss: 3.7547 - mean_absolute_error: 3.7547
[3.7547240257263184, 3.7547240257263184]
```

Figura 35: Evaluación VGG16 sin data augmentation

La métrica usada para evaluar la red ha sido el error absoluto medio. Vemos que el valor obtenido es 3.754.

5.4.2 VGG16 con data augmentation.

El resultado obtenido para la red VGG16 entrenada usando técnicas de data augmentation ha sido:

```
[ ] model.evaluate(test_images)

    4576/4576 [==============================] - 46s 10ms/step - loss: 4.7626 - mean_absolute_error: 4.7626
    [4.762634754180908, 4.762634754180908]
```

Figura 36: Evaluación VGG16 con data augmentation

La métrica usada para evaluar la red ha sido el error absoluto medio. Vemos que el valor obtenido es 4.762.

5.5 Análisis y discusión.

A continuación, vamos a analizar y comparar el resultado obtenido por las redes entrenadas en el proyecto.

En primer lugar, como hemos visto en el apartado anterior, el resultado obtenido por la red CNN propuesta sobre el conjunto de prueba sin usar técnicas de data augmentation ha sido 3.729. Si lo comparamos con el valor de pérdida en validación que se obtuvo para este modelo sin usar técnicas de data augmentation, puede observarse que el valor es cercano, lo que quiere decir que el modelo, en general, se comporta bien ante datos nuevos.

Igualmente, si nos fijamos en el resultado obtenido por la red CNN propuesta sobre el conjunto de prueba usando técnicas de data augmentation, vemos que el valor ha sido 3.827. Nuevamente, si lo comparamos con el valor de pérdida en validación que se obtuvo para este modelo usando técnicas de data augmentation, puede observarse que el valor es cercano, lo que quiere decir que el modelo, en general, se comporta bien ante datos nuevos. Si ahora lo comparamos con el caso anterior en el que no se usaron técnicas de data augmentation, puede verse que el resultado obtenido es ligeramente peor. Esto se verifica apoyándonos en la gráfica de las curvas de pérdida en entrenamiento y validación, analizado en el apartado 5.1.2, ya que se observaron variaciones grandes en los valores de pérdida entre una época y la siguiente, lo que parecía indicar que el modelo estaba generalizando peor.

Por otra parte, si nos fijamos en el resultado obtenido por la red VGG16 sobre el conjunto de prueba sin usar técnicas de data augmentation, vemos que el valor ha sido 3.754. Si lo comparamos con el valor de pérdida en validación que se obtuvo para este modelo sin usar técnicas de data augmentation, puede observarse que el valor es cercano, lo que quiere decir que el modelo, en general, se comporta bien ante datos nuevos.

Si ahora nos fijamos en el resultado obtenido por la red VGG16 sobre el conjunto de prueba usando técnicas de data augmentation, vemos que el valor ha sido 4.762. En este caso, aunque el resultado obtenido es peor que para el caso en el que se entrenó la red VGG16 sin usar data augmentation, también podemos ver que el modelo se comporta bien ante datos nuevos, ya que este valor es cercano al valor de pérdida en validación que se obtuvo para este modelo usando técnicas de data augmentation.

Realizando un análisis entre ambos modelos, podemos comprobar que la curva de pérdida en entrenamiento y validación parece ir estabilizándose con el paso del entrenamiento para el caso de la red VGG16, mientras que para la red CNN propuesta no se aprecia una estabilización, siendo el peor caso cuando se usa data augmentation, probablemente, provocado por el uso de un valor de learning rate grande. Tanto para la red CNN propuesta como para la red VGG16, comprobamos que al usar técnicas de data augmentation el resultado obtenido es peor que sin usar estas técnicas. Esto puede ser debido a que el modelo no sea capaz de extraer toda la información de las imágenes, además del valor del parámetro learning rate, como acabamos de comentar.

Por lo tanto, podemos afirmar que tanto la red CNN propuesta como la red VGG16 obtienen resultados similares.

6 Conclusión, valoración y futuros trabajos.

El objetivo de este TFM era utilizar todos los conocimientos adquiridos en las asignaturas de Big Data y Deep Learning para crear un sistema que reciba una fotografía de un panel fotovoltaico y de una predicción de la pérdida de potencia en la transformación de energía solar en energía eléctrica según el nivel de suciedad presente en la superficie del panel.

Durante este TFM, se han procesado y manipulado los datos necesarios para abordar el proyecto, como son las imágenes de los paneles fotovoltaicos y los datos numéricos de los valores de pérdida de potencia facilitados por los creadores del proyecto *Deep Solar Eye*. También se ha profundizado en los conocimientos de Deep Learning, al tener que crear y adaptar una red neuronal convolucional para resolver problemas de regresión, o incluso aprender nuevas técnicas como transfer learning, para aplicarlas a nuestro objetivo planteado.

Como valoración de este TFM, podemos afirmar que los objetivos han sido cumplidos, ya que se ha conseguido desarrollar un sistema que analiza una imagen de un panel fotovoltaico y da una predicción de la pérdida de potencia que provocaría la suciedad acumulada en la superficie del panel. La siguiente imagen es una muestra de ello, donde puede verse las predicciones hechas por la red VGG16 entrenada sin usar técnicas de data augmentation ante datos de prueba:

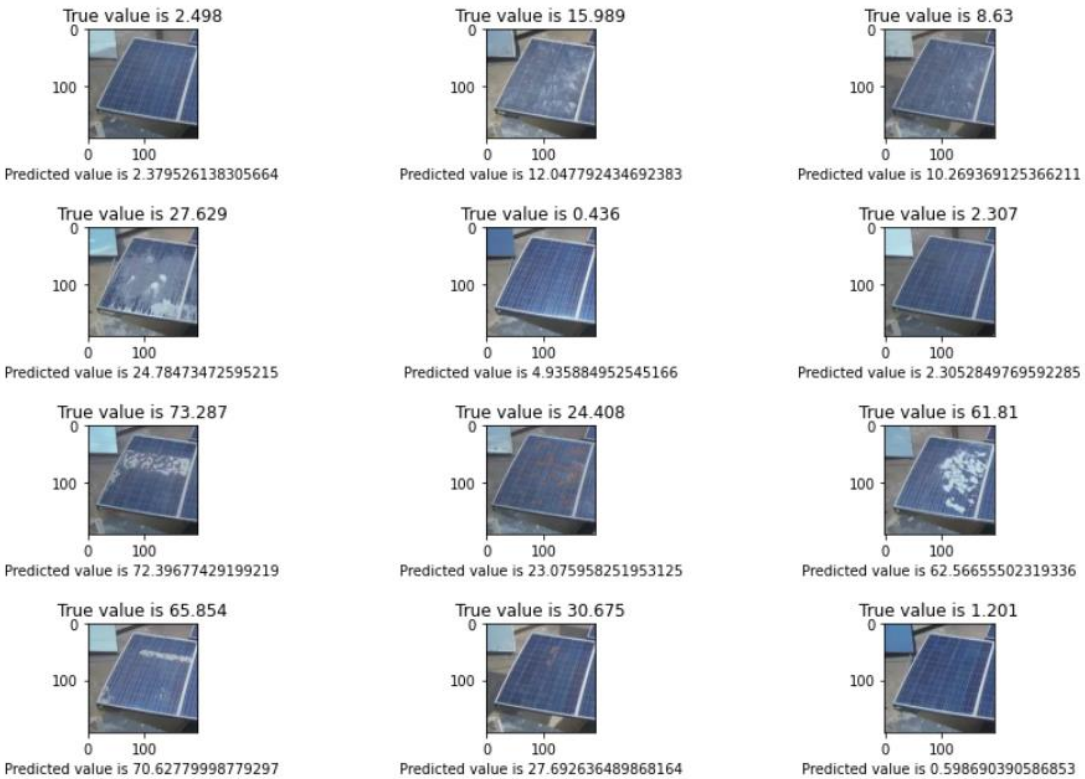


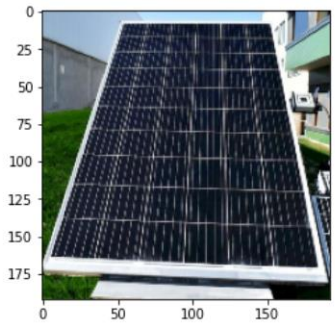
Figura 37: Predicciones VGG16

Por lo tanto, teniendo en cuenta los resultados obtenidos, se puede afirmar que tiene sentido utilizar imágenes para predecir la producción de paneles fotovoltaicos, ya que puede hacerse con unos niveles aceptables de precisión.

6.1 Futuros trabajos

Hemos podido comprobar que el sistema se comporta bien ante datos nuevos, siempre que las imágenes de los paneles fotovoltaicos sean del mismo estilo a las imágenes con las que el modelo ha sido entrenado.

Sin embargo, si intentamos hacer una predicción de una tipología de panel fotovoltaico diferente, comprobamos que el resultado no es el esperado:




```
[ ]  # Añadimos la dimensión perteneciente al Batch ya que hay que respetar el formato que espera la red
     img = np.expand_dims(norm_image, axis=0)

     # Hacemos la predicción
     model.predict(img)
```

```
1/1 [==============================] - 0s 18ms/step
array([[38.486256]], dtype=float32)
```

Figura 38: VGG16 predicción panel diferente

La parte positiva es que nuestro modelo es capaz de procesar imágenes completamente diferentes a las imágenes con las que ha sido entrenado, aunque la predicción que nos muestra es alta, 38.486, y no parece que el panel tenga suciedad superficial. Puede ser que las distintas líneas blancas que aparecen en la superficie del panel se estén tomando como suciedad.

Como futuros trabajos proponemos:

- Entrenar el modelo con un conjunto de datos más amplio, con diferentes tipologías de paneles fotovoltaicos, para conseguir un modelo más robusto.
- Usar otros modelos de redes neuronales convolucionales con el objetivo de mejorar los resultados obtenidos por los modelos usados en este proyecto.

REFERENCIAS

[1] Paneles fotovoltaicos: imagen.

https://www.aprean.com/wp-content/uploads/2020/12/Partes-de-un-panel-fotovoltaico.jpeg

[2] Células fotovoltaicas: información.

https://es.wikipedia.org/wiki/C%C3%A9lula_fotoel%C3%A9ctrica

[3] Paneles fotovoltaicos: información.

https://energyeducation.ca/Enciclopedia_de_Energia/index.php/C%C3%A9lula_fotovoltaica

[4] Deep Solar Eye.

https://www.arxiv-vanity.com/papers/1710.03811/

S. Mehta, A. P. Azad, S. A. Chemmengath, V. Raykar and S. Kalyanaraman,DeepSolarEye: Power Loss Prediction and Weakly Supervised Soiling Localization via Fully Convolutional Networks for Solar Panels," 2018 IEEE Winter Conference on Applications of Computer Vision (WACV), Lake Tahoe, NV, 2018, pp. 333-342.

[5] House Prices.

https://rosenfelder.ai/keras-regression-efficient-net/

[6] Book Prices.

https://python.plainenglish.io/judge-the-book-price-by-its-cover-with-image-regression-using-cnns-python-770707e4fe67

[7] Automatic detection of crohn disease in wireless capsule endoscopic images using a deep convolutional neural network.

https://link.springer.com/content/pdf/10.1007/s10489-022-04146-3.pdf

[8] VGG16: información e imágenes.

https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918#:~:text=What%20is%20VGG16%20used%20for,to%20use%20with%20transfer%20learning