PROYECTO FIN DE GRADO

TÍTULO: Corrección automática de imágenes subexpuestas y sobreexpuestas con técnicas de aprendizaje automático.

AUTOR/A: Patricia Ruiz Alonso

TITULACIÓN: Ingeniería de Sonido e Imagen

TUTOR/A: César Díaz Martín

DEPARTAMENTO: Ingeniería Audiovisual y Comunicaciones

VºBº TUTOR/A

Miembros del Tribunal Calificador:

PRESIDENTE/A: Rita Hogan Teves de Almeida

TUTOR/A: César Díaz Martín

SECRETARIO/A: David Luengo García

Fecha de lectura:

Calificación:

El Secretario/La Secretaria,

Agradecimientos

El camino desde el inicio de la carrera hasta la redacción de esta memoria ha sido complicado y ha requerido muchísimo esfuerzo, aunque hay ciertas personas que han ayudado a aliviar la carga, a las cuales me gustaría agradecer:

A mi tutor, César Díaz Martín, por haberme ayudado tanto durante el desarrollo de este proyecto, respondiendo a mis llamadas de auxilio incluso en vacaciones.

A mis padres, por hacerme sentir orgullosa del trabajo que ido llevando a cabo durante todos estos años, y por haber creído siempre en mí.

A mi hermano, por sacarme sonrisas con sus bromas y soportar mis frases de empollona.

A mis abuelos, a quienes más ilusión me hace poder decirles que voy a ser graduada en ingeniería.

A mi tío Diego y mi primo Alejandro, que siempre se han preocupado por mí.

A todos los amigos que he conocido en la escuela, que han conseguido que nunca me sienta sola, en especial a Antonio, Carlos, Sara, Andrés y Vera. No sé qué podría haber hecho sin todos vosotros.

A los miembros del CAT, la asociación de fotografía de la escuela, por inundarme de risas y entenderme cuando necesitaba una siesta.

A todos mis amigos de Alcalá, en especial a Marta, Paula y Nieves, que me han acompañado siempre.

A Sergi, que me ha dado todo el cariño del mundo, y ha estado conmigo en las malas y en las aún peores, y nunca ha dejado de apoyarme.

Resumen

Este proyecto, el cual recibe el título de “Corrección automática de imágenes subexpuestas y sobreexpuestas con técnicas de aprendizaje automático”, tiene como fin la creación de un modelo de aprendizaje automático propio para realizar la investigación de las técnicas y arquitecturas necesarias para llevar a cabo la corrección automática de errores de exposición tales como la subexposición y la sobreexposición. Este tipo de errores son muy comunes en el ámbito de la fotografía, y aunque su corrección es posible a través del uso de herramientas y aplicaciones externas, la realización de estas modificaciones y el manejo de dichos recursos resulta complejo para un usuario no especializado. Por todo ello, el proyecto realiza el estudio de un modelo de aprendizaje automático capaz de llevar a cabo la tarea de corrección de errores de exposición, donde se busca la sustitución total de la intervención del usuario.

El estado del arte presenta diversos sistemas como solución al problema propuesto, destacando algunos con resultados muy positivos. Sin embargo, los modelos existentes utilizan arquitecturas bastante complejas y elaboradas. En este proyecto se busca el diseño de una solución más sencilla pero ambiciosa, donde se pretenden alcanzar resultados comparables con los ofrecidos en las publicaciones del estado del arte utilizadas como referencia.

Para la realización de la solución propuesta se utiliza una arquitectura de red tipo U-net convolucional contractiva, a través del desarrollo de código fuente en el lenguaje de programación Python. De esta manera se utilizan diversas librerías especializadas en el diseño de modelos de aprendizaje automático como Tensorflow y Keras, junto a librerías utilizadas para el procesado de imágenes como Scikit-image o Matplotlib. También se desarrollan funciones propias para poder realizar diferentes labores, necesarias para el correcto funcionamiento del modelo. Se proponen así una función de pérdida contractiva propia, un sistema de adaptación de las imágenes de entrada y una aplicación de técnicas de *weight clipping*, entre otras.

El desarrollo de la red pasa así por cuatro etapas. En la primera de ellas se comienza con la documentación, junto a la implementación de la arquitectura inicial y de la estructura del código fuente. Tras ello se realiza la adaptación y preprocesado de las imágenes de entrada, provenientes de la base de datos escogida. Después, se lleva a cabo el entrenamiento y validación del modelo, etapa en la que se realizan diversos entrenamientos de prueba. Finalmente se conduce la etapa de pruebas para obtener las imágenes predichas por el modelo implementado. Cabe destacar que se sigue un modelo de trabajo iterativo, por lo que la arquitectura final de la red es desarrollada a través de una serie de iteraciones en función de los resultados obtenidos en los entrenamientos de prueba y en la etapa de pruebas.

En esta memoria se procede a mostrar, explicar y analizar los resultados de los entrenamientos de prueba y de sus respectivas imágenes resultantes, para comprobar cuál es el efecto de las técnicas y de los hiperparámetros que son utilizados a lo largo del desarrollo del modelo. Esto se refleja en las métricas aplicadas, donde se utilizan las impuestas por el estado del arte, junto a otras que se han considerado de importancia para medir el desempeño de la red.

Finalmente, los resultados alcanzados concluyen en la creación de un modelo que puede considerarse como base para la implementación de un modelo más avanzado, capaz de llevar a cabo el objetivo final. Se ofrecen así direcciones futuras para poder desarrollar la red especificada, en función de los comportamientos analizados a lo largo del desarrollo de las pruebas.

Este proyecto supone además un impacto positivo en la actualidad, debido a la investigación que proporciona y a sus futuras contribuciones en el ámbito social y medioambiental, detallando también su aportación a los Objetivos de Desarrollo Sostenible.

Abstract

This project, titled "Automatic Correction of Underexposed and Overexposed Images Using Machine Learning Techniques," aims to develop a custom machine learning model to explore the techniques and architectures required for the automatic correction of exposure errors such as underexposure and overexposure. These types of errors are common in photography, and while they can be corrected using external tools and applications, doing so can be complex for non-specialized users. Therefore, this project focuses on developing a machine learning model capable of performing exposure error corrections autonomously, eliminating the need for user intervention.

The state of the art presents various systems as solutions to the proposed problem, with some achieving very positive results. However, existing models often rely on complex and elaborate architectures. This project seeks to design a simpler yet ambitious solution, aiming to achieve results comparable to those presented in state-of-the-art references.

The proposed solution utilizes a U-Net convolutional contractive architecture, developed using Python. The project employs specialized libraries for designing machine learning models, such as TensorFlow and Keras, alongside image processing libraries like Scikit-image and Matplotlib. Additionally, custom functions are developed to perform various tasks which are essential for the model's proper functioning. These include a custom contractive loss function, an input image adaptation system, and the application of weight clipping techniques, among others.

The development of the network progresses through four stages. The first stage involves documentation, as well as the implementation of the initial architecture and source code structure. Next, the input images, sourced from the selected database, are adapted and preprocessed. Following this, the model goes through training and validation stages, during which multiple test trainings are conducted. Finally, the testing stage is carried out to obtain the images predicted by the implemented model. An iterative work model is followed, so the final network architecture is developed through a series of iterations based on the results obtained from test trainings and the testing phase.

This report presents, explains, and analyzes the results of the test trainings and the corresponding resulting images to assess the impact of the techniques and hyperparameters used throughout the model's development. This is reflected in the applied metrics, which include those established by the state of the art, as well as others considered important for measuring the network's performance.

Ultimately, the results achieved lead to the creation of a model that can serve as a foundation for implementing a more advanced model capable of fulfilling the final objective. Future directions for further developing the specified network are also provided, based on the behaviors analyzed during the training phase and the testing phase.

Moreover, this project has a positive impact due to the research it provides and its potential future contributions in the social and environmental spheres, aligning with the Sustainable Development Goals.

Índice de figuras

Figura 1. Triángulo de exposición, extraído de [1] 5

Figura 2. Izquierda: Imagen subexpuesta. Centro: Imagen con exposición correcta. Derecha: Imagen sobreexpuesta [2] 6

Figura 3: Estructura de una neurona 7

Figura 4: Estructura de las capas en una red neuronal, extraído de [4] 7

Figura 5: Función de activación sigmoide, extraído de [4] 8

Figura 6: Función de activación ReLU, extraído de [4] 9

Figura 7: Visión general de la estructura de un modelo de CNN, extraído de [6] 10

Figura 8: Ejemplo de una convolución de una imagen de dimensiones 3x3 en 2D con un filtro de 2x2 y *stride* de valor 1, extraído de [8] 11

Figura 9: Ejemplo de *padding* en una imagen 2D con tamaño original de 4x4, extraído de [8] 12

Figura 10: Ejemplo de *max pooling* utilizando un filtro de 2x2, con un *stride* de valor 2, extraído de [8] 12

Figura 11: Ejemplo resumido de una CNN, extraído de [8] 13

Figura 12: Arriba: Estructura general de un CAE, extraído de [11]. Abajo: Ejemplo de un CAE utilizado para la eliminación de ruido, extraído de [10] 14

Figura 13: Estructura de una U-Net, extraído de [12] 15

Figura 14: Estructura general del modelo de Afifi et al. [2] 18

Figura 15: Estructura del sistema propuesto por [17] 19

Figura 16: Esquema general del modelo. 23

Figura 17: Diagrama de la estructura final del modelo 24

Figura 18: Ejemplos de imágenes del *dataset* [2]. De izquierda a derecha, con valores de EV relativo respecto a su original de -1’5, -1, +0, +1, +1’5 26

Figura 19: Izquierda: Imagen de entrada con valor de EV relativo de +0. Derecha: Imagen *ground truth*, retocada por “*Expert C*” [30] 26

Figura 20: Árbol de directorios con la organización de las imágenes del *dataset* 27

Figura 21: Esquema del proceso que siguen las imágenes de entrada 28

Figura 22: Esquema de la organización de las imágenes 29

Figura 23: Esquema del entrenamiento del modelo 30

Figura 24: Resultados del entrenamiento de la prueba 4, con $\lambda = 100$ y un *learning rate* de 10^{-4}. Arriba: Valor de la pérdida. Abajo: Valor de las métricas PSNR y SSIM 34

Figura 25: Resultados del entrenamiento de la prueba 6, con $\lambda = 0{,}1$ y un *learning rate* de 10^{-3}. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM 36

Figura 26: Imágenes predichas por los modelos de las pruebas 6 y 11. Arriba: Representación real de las imágenes. Abajo: Representación en función de los valores mínimos y máximos de los píxeles 37

Figura 27: Imagen *ground truth* de las imágenes predichas 38

Figura 28: Resultados del entrenamiento de la prueba 7, con λ = 0,1 y un *learning rate* de 10^{-2}. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM 39

Figura 29: Comparativa de los resultados de subida repentina de la pérdida. Arriba izquierda: Prueba 7. Arriba derecha: Prueba 8. Abajo izquierda: Prueba 10. Abajo derecha: Prueba 12 40

Figura 30: Resultados del entrenamiento de la prueba 15, con λ = 0,01 y un *learning rate* de 10^{-3}. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM 41

Figura 31: Imágenes predichas por el modelo de la prueba 15. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles..... 42

Figura 32: Resultados del entrenamiento de la prueba 16, con λ = 0,01, un learning rate de 10^{-3} y *weight clipping* de ±0,1. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM 44

Figura 33: Resultados del entrenamiento de la prueba 17, con λ = 0,01, un learning rate de 10^{-3} y *weight clipping* de ±0,05. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM 46

Figura 34: Imágenes predichas por el modelo de la prueba 17. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles..... 47

Figura 35: Resultados del entrenamiento de la prueba 19, con λ = 0,01 y un learning rate de 10^{-3}. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM 48

Figura 36: Imágenes predichas por el modelo de la prueba 19. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles..... 49

Figura 37: Otras imágenes predichas por el modelo de la prueba 19, representadas en función de sus valores mínimos y máximos 50

Figura 38: Resultados del entrenamiento de la prueba 21, con λ = 0,01, un learning rate de 10^{-4} y *clipnorm* de 0,5. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM 52

Figura 39: Imágenes predichas por el modelo de la prueba 21. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles..... 53

Figura 40: Resultados del entrenamiento de la prueba 22, con λ = 0,01, un learning rate de 10^{-4} y *clipnorm* de 1. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM 54

Figura 41: Imágenes predichas por el modelo de la prueba 22. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles..... 55

Figura 42: Resultados del entrenamiento de la prueba 23, con λ = 0,01, un learning rate de 10^{-4} y *clipvalue* de 5. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM 56

Figura 43: Imágenes predichas por el modelo de la prueba 22. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles..... 57

Figura 44: Reproducción de *Table 1* de [17], resaltando con un rectángulo rojo los resultados de interés. Esta tabla contiene y amplía los resultados de [2]..... 58

Índice de tablas

Tabla 1: Resultados destacados de las métricas utilizando un modelo simple 35
Tabla 2: Resultados de la evaluación de los valores de los modelos de las pruebas 6 y 11 36
Tabla 3: Resultados de la evaluación de los valores del modelo de la prueba 15 41
Tabla 4: Resultados de la evaluación de los valores del modelo de la prueba 16 45
Tabla 5: Resultados de la evaluación de los valores del modelo de la prueba 17 46
Tabla 6: Resultados de la evaluación de los valores del modelo de la prueba 19 48
Tabla 7: Resultados de la evaluación de los valores del modelo de la prueba 21 52
Tabla 8: Resultados de la evaluación de los valores del modelo de la prueba 22 54
Tabla 9: Resultados de la evaluación de los valores del modelo de la prueba 23 57
Tabla 10: Presupuesto del proyecto 59

Lista de acrónimos

ANN: *Artificial Neural Network*

CAE: *Convolutional Autoencoder*

CNN: *Convolutional Neural Network*

EV: *Exposure Value*

GT: *Ground Truth*

MS-SSIM: *Multi-Scale Structural Similarity Index Measure*

MSE: *Mean Squared Error*

ODS: Objetivos de Desarrollo Sostenible

PSNR: *Peak Signal-to-Noise Ratio*

ReLU: *Rectified Linear Unit*

SSIM: *Structural Similarity Index Measure*

SVH: Sistema Visual Humano

Índice de contenidos

Agradecimientos ..... **i**

Resumen ..... **iii**

Abstract ..... **v**

Índice de figuras ..... **vii**

Índice de tablas ..... **ix**

Lista de acrónimos ..... **xi**

1. Introducción **1**

1.1. Marco y motivación del proyecto 1

1.2. Objetivos técnicos y académicos 2

1.3. Estructura del resto de la memoria 2

2. Marco tecnológico **5**

2.1. Sobreexposición y subexposición 5

2.2. Redes neuronales y modelos de aprendizaje automático 6

2.2.1. Estructura 6

2.2.2. Entrenamiento 7

2.2.3. Funciones de activación 8

2.3. Redes Neuronales Convolucionales CNN 9

2.3.1. Etapa de extracción de características 10

2.3.2. Etapa de clasificación 13

2.3.3. Entrenamiento 13

2.4. Autoencoders convolucionales 13

2.4.1. U-Net 15

2.4.2. Autoencoders contractivos 15

2.5. Métricas 16

2.5.1. PSNR 16

2.5.2. SSIM 17

2.5.3. MS-SSIM 17

2.6. Trabajos previos y estado del arte 18

3. Especificaciones y restricciones de diseño **21**

4. Descripción de la solución propuesta **23**

4.1. Estructura 24

4.2. Entradas 25

4.2.1. Base de datos 25

4.2.2. Preprocesado e introducción de las imágenes en el modelo 27

4.3. Entrenamiento 29

4.3.1. Función de pérdida 30

4.3.2. Hiperparámetros 31

4.3.3. Métricas 31

4.4. Etapa de pruebas 32

5. Resultados 33

5.1. Primeros entrenamientos con un modelo simple 33

5.1.1. Resultados con un valor alto de lambda 33

5.1.2. Resultados variando el valor de lambda 34

5.2. Pruebas modificando la arquitectura del modelo 40

5.3. Pruebas con técnicas de *clipping* 42

5.3.1. Primera técnica: *weight clipping* 43

5.3.2. Segunda técnica: *clipnorm* 50

5.3.3. Tercera técnica: *clipvalue* 55

5.4. Discusión final 57

6. Presupuesto 59

7. Impacto del proyecto 61

8. Conclusiones 63

8.1. Conclusiones 63

8.2. Trabajos futuros 63

9. Referencias 65

Anexos 69

A.1 Función de adaptación de los datos de salida 69

A.2 Función de pérdida 69

A.3 Clase *WeightClippingCallback* 70

1. Introducción

1.1. Marco y motivación del proyecto

En la actualidad, la edición fotográfica es llevada a cabo de manera regular en diversos contextos, desde de manera casual enfocada en el ocio y el ámbito social, hasta de manera profesional, utilizada por fotógrafos expertos para realizar cambios sutiles o corregir errores surgidos al tomar la fotografía. En relación con dichos errores, hay parámetros que resultan complicados de ajustar a la hora de utilizar cámaras fotográficas, incluso si estos se configuran de manera automática. De esta manera, uno de los errores más comunes en la fotografía es la toma errónea de la luz, que da lugar a errores de exposición. Estos pueden ocurrir debido a dos fenómenos, la subexposición o a la sobreexposición.

Hoy en día existen múltiples programas para la edición y corrección fotográfica, algunos de ellos muy conocidos, tales como Adobe Photoshop. Este tipo de programas incluyen además opciones avanzadas con las que obtener muy buenos resultados, sin embargo, pueden resultar complicados y complejos para un usuario no especializado. Además, no todas las fotografías pueden ser fácilmente retocadas para corregir errores de exposición.

Tomando una aproximación técnica, el procesado de imagen ha conseguido en los últimos años un gran auge gracias al desarrollo de modelos de aprendizaje automático y profundo, logrando con ello un avance y una mejora considerable que continúa en el presente. Gran parte de estos avances tienen su base en las redes neuronales artificiales, desde las cuales se han desarrollado otros modelos más adecuados para el procesado de imágenes como las redes neuronales convolucionales, conocidas como CNN (Convolutional Neural Network). Este tipo de redes, utilizadas con estructuras variadas, son muy versátiles y aparecen en muchos sistemas tales como modelos para la eliminación de ruido y restauración de imágenes, o para la detección, reconocimiento y clasificación de objetos, entre otras funcionalidades. Esto ha resultado muy interesante en campos como la medicina, para mejorar la obtención de diagnósticos, a través del procesado de imágenes médicas. También destacan otro tipo de aplicaciones como la segmentación semántica, o el reconocimiento facial y de emociones.

Teniendo esto en cuenta, se busca llevar a cabo un modelo de aprendizaje automático profundo que sea capaz de corregir imágenes subexpuestas y sobreexpuestas de manera automática, con una intervención mínima del usuario. De esta manera, uno de los objetivos reside en que el modelo pueda ser utilizado por cualquier tipo de usuario que necesite realizar un retoque fotográfico para editar la iluminación de una imagen, sin necesidad de experiencia previa o de recurrir a programas externos. Está así principalmente dirigido a un público amante de la fotografía, pero podría ser utilizado sin problema en otro tipo de escenarios donde la finalidad buscada sea realizar un procesado de la imagen respecto a su exposición, para aumentar así los datos que se podrían obtener de esta.

Para ello se propone el uso de una CNN, concretamente a través de un autoencoder tipo U-Net convolucional contractivo. El estado del arte presenta diversos modelos de redes profundas que tratan la corrección de la exposición, fundamentados en arquitecturas similares, pero que incorporan un mayor número de bloques y presentan una mayor complejidad. Se busca con este proyecto alcanzar el desarrollo de un modelo más sencillo que las soluciones empleadas en el estado del arte, con el que se puedan alcanzar valores y resultados comparables con dichas publicaciones.

1.2. Objetivos técnicos y académicos

Este proyecto presenta los siguientes objetivos técnicos:

- Desarrollo de código fuente en el lenguaje de programación Python, utilizando las bibliotecas correspondientes, para llevar a cabo un modelo de aprendizaje automático propio, que siga la estructura de un autoencoder convolucional contractivo. Dicho código fuente no debe presentar errores de ejecución.
- Búsqueda o creación de una base de datos adecuada respecto al problema a tratar, con la que se pueda realizar el entrenamiento del modelo, junto a la validación y pruebas posteriores.
- Realización de un preprocesado y tratamiento de los datos previo al modelo.
- Diseño y desarrollo de funciones auxiliares propias, necesarias para la puesta en marcha del modelo.
- Realización de un diseño de experimentos para la comprobación de la arquitectura y ajuste del modelo
- A través de los resultados obtenidos, encontrar los hiperparámetros, función de pérdida y métricas más adecuados para conseguir y demostrar el adecuado funcionamiento del modelo.

Desde el punto de vista académico, la alumna desarrolla a lo largo de la realización del proyecto diversas competencias y habilidades mencionadas a continuación:

- Aprendizaje y comprensión del lenguaje de programación Python y sus arquitecturas.
- Aprendizaje del alcance de las librerías TensorFlow, Keras o Scikit.io, entre otras. Desarrollo de código en Python, en relación con el uso dichas librerías.
- Comprensión de las estructuras de las redes neuronales convolucionales y autoencoders, profundizando en las U-Nets.
- Habilidad para trabajar con sistemas de procesado de señal, en especial de procesado digital de imagen.
- Capacidad para la búsqueda de fuentes de información adecuadas a través del trabajo autónomo.
- Capacidad de organización para el desarrollo de un proyecto avanzado en tiempo limitado.

1.3. Estructura del resto de la memoria

La memoria del proyecto se estructura en diferentes capítulos, descritos a continuación:

1. Introducción
2. Marco tecnológico: Se describe el contexto en el cual se encuentra el proyecto, comenzando con el funcionamiento de las redes neuronales y redes neuronales convolucionales. Tras ello se explican las arquitecturas de los autoencoders, profundizando en las redes tipo U-Net. Se finaliza con la introducción a las métricas utilizadas en el proyecto, y a los modelos correspondientes al estado del arte.
3. Especificaciones y restricciones de diseño: Se establecen los requisitos que el proyecto debe cumplir, junto a las limitaciones que se pueden encontrar en su diseño.
4. Descripción de la solución propuesta: Se presenta el modelo desarrollado, junto a las etapas que recorre este para poder realizar las pruebas. Se explican las funciones propias y auxiliares que la red utiliza en cada uno de los procesos realizados.

5. Resultados: Se presentan las pruebas realizadas para comprobar el funcionamiento de la red, mientras se razonan y analizan los resultados en función de la arquitectura y técnicas utilizadas. En este capítulo se muestra la metodología de trabajo iterativo para el desarrollo del modelo final.
6. Presupuesto: Se ofrece el presupuesto necesario para el desarrollo del proyecto.
7. Impacto del proyecto: Se analiza el impacto que supone la solución propuesta, junto con su implicación con los Objetivos de Desarrollo Sostenible.
8. Conclusiones: Se concluye el resultado final del proyecto, incluyendo las líneas futuras de trabajo en función de la solución obtenida.
9. Referencias.

2. Marco tecnológico

2.1. Sobreexposición y subexposición

En el ámbito de la fotografía, la luz supone una parte fundamental a la hora de capturar una imagen. Esta regula el nivel de exposición de la fotografía tomada, y en una cámara digital se puede controlar a través de tres parámetros, conocidos como el triángulo de exposición. El triángulo de exposición indica el equilibrio entre la apertura del diafragma, la sensibilidad ISO y la velocidad de obturación, y se expone en la Figura 1. Estos elementos se pueden controlar manualmente con el modo manual en una cámara, o automáticamente con el modo automático, donde este último busca los valores de los parámetros que consigan un mejor equilibrio entre los tres según la escena fotografiada, para lograr un valor de exposición correcto. Sin embargo, el equilibrio entre dichos parámetros es algo complejo de conseguir en modo manual, y puede fallar incluso en modo automático, en especial si hay presencia de condiciones complejas de iluminación como situaciones de contraluz o de luz escasa, dando así lugar a fenómenos como la subexposición y la sobreexposición.

Figura 1. Triángulo de exposición, extraído de [1]

Tal como su nombre indica, la subexposición y sobreexposición son niveles inferiores y superiores al nivel correcto de exposición en una imagen respectivamente. Estos fenómenos son también conocidos en el ámbito de la fotografía como fotografías oscuras en el caso de imágenes subexpuestas, o fotografías quemadas para las imágenes sobreexpuestas. De esta manera, una imagen subexpuesta supone una fotografía en la cual la cantidad de luz que ha sido capturada está por debajo de la cantidad de luz necesaria, por lo que esta se aprecia oscura o con zonas negras a simple vista, mientras que en una imagen sobreexpuesta la cantidad de luz capturada se encuentra por encima del nivel de luz necesario, y por ello se aprecia muy clara o con zonas blancas. En la Figura 2 se muestra un ejemplo de una misma fotografía con diferentes valores de exposición aplicados, de manera que se aprecia visualmente la diferencia entre los diferentes tipos de exposición.

Figura 2. Izquierda: Imagen subexpuesta. Centro: Imagen con exposición correcta. Derecha: Imagen sobreexpuesta [2]

La exposición se puede medir de diferentes maneras, siendo una de ellas el Valor de Exposición EV (Exposure Value). Este relaciona la apertura del diafragma y el tiempo de exposición utilizados al tomar una fotografía, ofreciendo así diferentes valores de EV para diferentes combinaciones de la apertura del diafragma y del tiempo de exposición. Su definición formal sigue la expresión (1), donde N es la apertura del diafragma, expresada con el número f, y t es el tiempo de exposición en segundos.

$$EV = \log_2\left(\frac{N^2}{t}\right) \approx 3{,}32 \cdot \log_{10}\left(\frac{N^2}{t}\right) \quad (1)$$

De esta manera, EV se expresa en pasos, donde cada paso equivale a un incremento o decremento de 1 unidad de EV (±1 EV). Un paso ascendente implica duplicar el valor de exposición, de forma que se duplica la luz que afecta a la imagen, mientras que un paso descendente supone reducir a la mitad el valor de exposición, de manera que se reduce a la mitad la cantidad de luz que afecta a la imagen. Los diferentes valores de EV según distintas combinaciones de la apertura del diafragma y el tiempo de exposición se pueden encontrar en la Tabla de valores de exposición en [3]. Estos valores varían desde EV -6 que aplica a escenas muy oscuras, de aurora boreal medianamente luminosa, hasta EV 16 que aplica a escenas muy luminosas, con arena clara (playas) o nieve en plena luz.

2.2. Redes neuronales y modelos de aprendizaje automático

Las redes neuronales artificiales o ANN (Artificial Neural Networks) son un conjunto de algoritmos que tratan de imitar el funcionamiento del cerebro humano, simulando así conexiones de neuronas de manera artificial. Estos sistemas han demostrado muy buenos resultados en muchos campos y aplicaciones, debido a su capacidad para aprender datos y reconocer patrones, y hoy en día son la base del aprendizaje automático y de la inteligencia artificial.

2.2.1. Estructura

La unidad más básica de una red neuronal es una neurona artificial, también conocida como nodo. Cada neurona actúa como una unidad de procesamiento, la cual aplica una operación matemática para obtener una salida a través de una serie de entradas. La salida es calculada en función de las entradas, aplicando a cada una de estas un peso (*weight*), y sumando su valor. A esto se le añade un sesgo, también conocido como *bias*, y se le aplica una función de activación, para poder conseguir salidas no lineales. En la Figura 3 se aprecian todos los elementos presentes. Cada neurona lleva a cabo una operación simple, en la cual esta se activa si la señal total recibida excede el umbral de activación [4].

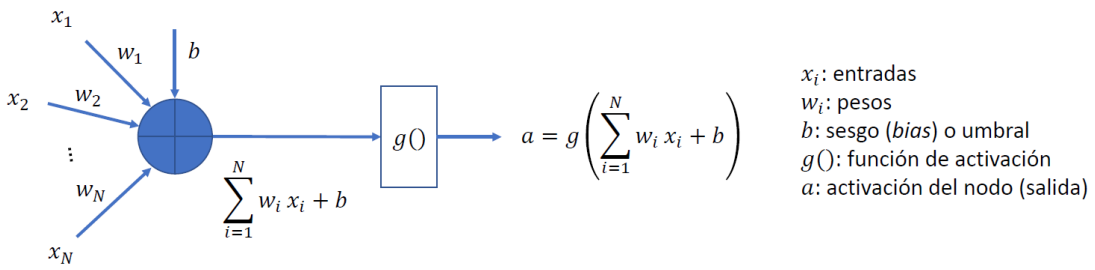

Figura 3: Estructura de una neurona

Estas neuronas se agrupan en capas, las cuales están conectadas con otras capas formando así una red neuronal. Las salidas de las neuronas de una capa se conectan con las entradas de las neuronas de la siguiente capa, organizándose así con una capa de entrada (*input*), una capa de salida (*output*), y una o más capas intermedias (*hidden layers*), que aumentan cuanto más complejo sea el modelo, formando la estructura que se observa en la Figura 4. El objetivo de las capas ocultas es conseguir una salida correcta a partir de una entrada determinada, a través del ajuste de sus pesos y sesgos y del uso de diferentes funciones de activación. En estas capas se consiguen obtener las características latentes de los datos de entrada, que son el resultado de la combinación no lineal de las características observables.

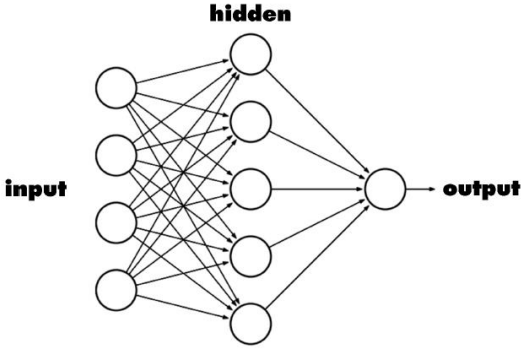

Figura 4: Estructura de las capas en una red neuronal, extraído de [4]

2.2.2. Entrenamiento

Para realizar el ajuste de los pesos y de los sesgos se entrena el modelo en la llamada etapa de entrenamiento, en la cual se ofrece al modelo una serie de datos de entrada a partir de los cuales se modifican los valores de los sesgos y de los pesos hasta que las salidas se aproximan a la función deseada. Para el entrenamiento existen tanto el aprendizaje supervisado como el no supervisado, los cuales dependen de si se entrega o no al modelo la salida correspondiente a una entrada. En el aprendizaje supervisado se entregan al modelo conjuntos de entradas con sus respectivas salidas esperadas, para que este pueda realizar los ajustes correspondientes en los parámetros mencionados. Estas salidas esperadas se conocen como *ground truth* (GT), y los ajustes se calculan a través de métodos de optimización como el descenso de gradiente [5]. Este método se encarga de minimizar el valor de las funciones de pérdida, también llamadas funciones de coste o *loss functions*, que calculan el error en la predicción del modelo. El objetivo del descenso de gradiente es ajustar los pesos de la red neuronal, según los valores de error obtenidos con las funciones de pérdida. Una de las funciones de pérdida más comunes para calcular el error entre la predicción generada por el modelo y el *ground truth* es el Error Cuadrático Medio MSE (*Mean Squared Error*), aunque la elección de la función de coste depende del modelo a tratar. Se expone el cálculo del MSE en la expresión (2), donde y_i es el

valor del *ground truth* e $\hat{y}_i$el valor de la salida predicho por el modelo, y m es el número de salidas en el conjunto de datos.

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i) \quad (2)$$

La etapa de entrenamiento se divide en épocas, también conocidas como *epochs*, en las cuales se realiza una iteración completa de todo el modelo donde se ajustan los parámetros utilizando uno o varios conjuntos de datos. Estos conjuntos o grupos de datos se llaman *batches*. Cuantas más épocas tenga el entrenamiento mejor entrenado estará el modelo, pero también habrá mayor riesgo de que ocurra un sobreajuste (*overfitting*). El sobreajuste es un fenómeno en el cual el modelo se sobreentrena, de manera que se adapta tanto a las características de los datos de entrenamiento que al introducir nuevos conjuntos de datos el modelo no es capaz de realizar buenas predicciones. Por ello, el sobreajuste siempre se debe tratar de prevenir, existiendo para ello diferentes prácticas, comenzando con evitar el sobreentreno del modelo.

2.2.3. Funciones de activación

Las funciones de activación sirven para conseguir que el procesamiento de los datos a través de las neuronas no sea lineal. Los problemas que las redes neuronales tratan de resolver son complejos y no lineales por naturaleza, y por ello se introducen las funciones de activación a la salida de cada neurona. Existen múltiples tipos de funciones de activación, pero se van a presentar algunas que son utilizadas en este proyecto [4].

2.2.3.1. Sigmoide

La función sigmoide es una de las funciones de activación más utilizadas, la cual sigue una curva en forma de *S* que se presenta en la Figura 5. Esta comprime las entradas de manera que las salidas que ofrece se encuentran entre el valor 0 y 1, usada por ello muchas veces en la clasificación binaria. Su expresión se observa en (3). Típicamente se observa en la última capa, para realizar la clasificación de la salida.

$$f(x) = \frac{1}{1 + e^{-x}} \quad (3)$$

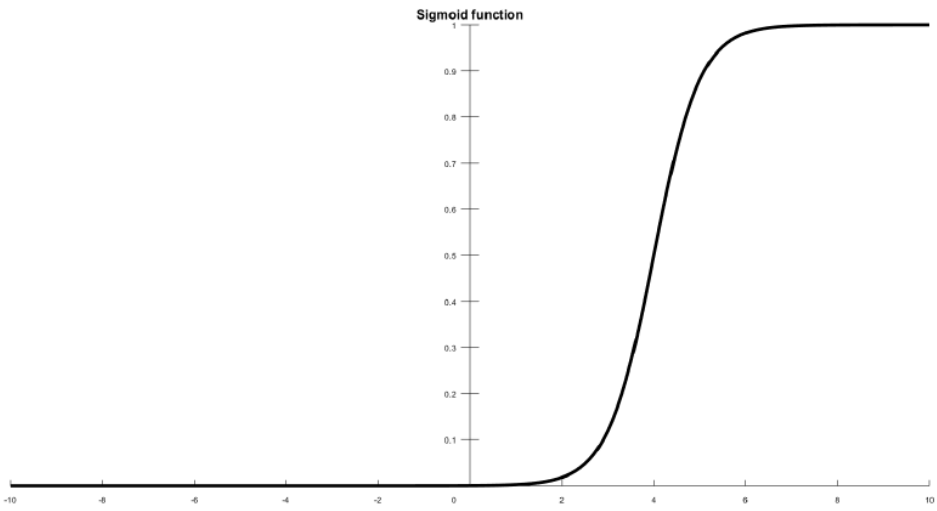

Figura 5: Función de activación sigmoide, extraído de [4]

2.2.3.2. ReLU (*Rectified Linear Unit*)

Es una función más simple que la anterior, pero también una de las más utilizadas, en especial en las capas intermedias (*hidden* layers) del modelo. Esta sigue la expresión mostrada en (4), de manera que su salida se encuentra en un rango entre 0 y el valor de la entrada. Una de sus aplicaciones más importantes se encuentra en la visión artificial, ya que es un factor clave en las CNN. En la Figura 6 se puede apreciar su forma.

$$f(x) = \max(0, x) \tag{4}$$

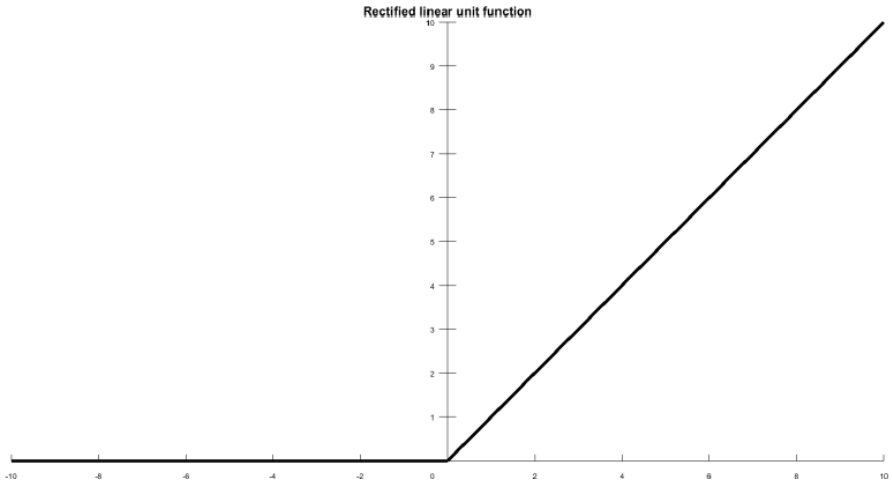

Figura 6: Función de activación ReLU, extraído de [4]

2.3. Redes Neuronales Convolucionales CNN

Para el procesado de imágenes con modelos de aprendizaje automático en la actualidad se utilizan las Redes Neuronales Convolucionales, abreviado como CNN. Este tipo de redes tienen un rendimiento mucho mayor en comparación a las redes neuronales tradicionales a la hora de trabajar con imágenes, mientras que su complejidad es mucho menor. Las CNN están diseñadas para tratar imágenes como argumento de entrada, consiguiendo captar la información que estas transmiten a partir de procesos de convolución. De esta manera, son uno de los recursos más utilizados en modelos de visión artificial al ser capaces de encontrar patrones y analizarlos para distintas tareas posteriores, tales como la clasificación de imágenes, la segmentación, o la detección de objetos, entre otras muchas aplicaciones. Algunas de sus salidas más destacadas se encuentran en el campo de la medicina, utilizado para el análisis de imágenes médicas, ya que se han desarrollado modelos que han demostrado su eficacia al facilitar la detección de tumores y la obtención de diagnósticos más acertados. Otra de las aplicaciones que destaca es su uso en vehículos autónomos, realizando análisis de las señales, el tráfico y los peatones para poder llevar a cabo una conducción autónoma. Se pueden encontrar muchos modelos también dedicados al reconocimiento facial, identificando a personas y clasificando emociones, y en el reconocimiento de la escritura a mano, siendo así capaz de realizar una digitalización de documentos.

Los modelos de CNN presentan una estructura general similar a las redes neuronales tradicionales, pero emplean capas intermedias diferentes, diseñadas específicamente para el procesamiento de imágenes. Aunque la estructura de diferentes modelos puede variar, existen tres tipos de capas que típicamente siempre se hayan presentes en una arquitectura de un modelo de CNN: las capas

convolucionales, las capas de pooling y las capas completamente conectadas. Además, la estructura de estos modelos se divide normalmente en dos grandes etapas: la etapa de extracción de características y la etapa de decisión o de clasificación, aunque esta última puede variar en función de la misión del modelo. Ambas se describen a continuación. En la Figura 7 se ofrece una visión general de la estructura mencionada.

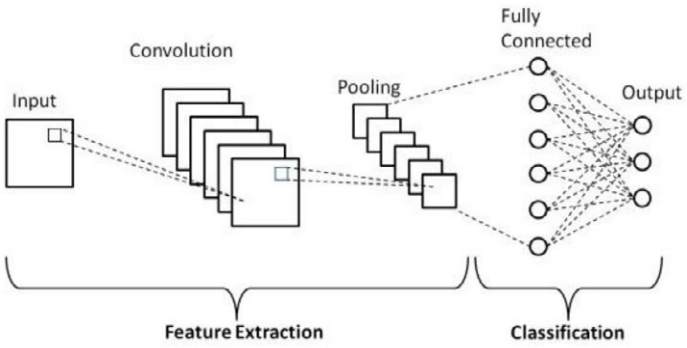

Figura 7: Visión general de la estructura de un modelo de CNN, extraído de [6]

2.3.1. Etapa de extracción de características

En una red neuronal tradicional, los pesos y sesgos utilizados para la clasificación pertenecen a neuronas individuales. Sin embargo, para analizar una zona de una imagen y obtener sus características no se puede utilizar el valor de cada píxel de manera independiente, sino que se debe tener en cuenta la relación espacial entre píxeles vecinos, debido a la alta correlación que hay entre ellos. Al estudiar un píxel junto a su contexto (sus píxeles vecinos) se puede conseguir extraer sus características, por ejemplo, si se trata de un borde o de una zona homogénea, entre otras. Para estudiar dicho contexto, se utilizan capas convolucionales.

2.3.1.1. Capas convolucionales

Las capas convolucionales, como su propio nombre indica, se encargan de realizar convoluciones a través del uso de filtros, también conocidos como *kernels*. Estos son pequeñas matrices, típicamente de tamaño de 3x3 píxeles y con n dimensiones, donde cada una de sus posiciones tiene un peso. Los filtros se encargan de recorrer toda la imagen, donde el filtro se desplaza sobre los píxeles con un paso determinado por un parámetro llamado *stride*, realizando la convolución discreta entre estos y los píxeles que forman la imagen. Las salidas de la operación quedan almacenadas en una nueva imagen llamada mapa de características (*feature map*). En este mapa de características se mostrarán diferentes resultados en función de los pesos del filtro, pudiendo así encontrar pequeños atributos o propiedades de la imagen, como bordes o cambios de textura, al utilizar los filtros adecuados para ello. En la Figura 8 se muestra un ejemplo de una convolución utilizando un filtro de n = 2 dimensiones, mientras que se puede encontrar un ejemplo visual de la convolución utilizando un filtro de mayores dimensiones en [7].

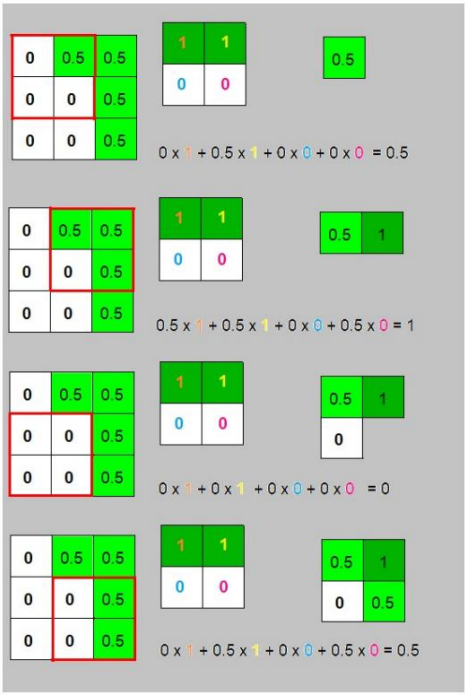

Figura 8: Ejemplo de una convolución de una imagen de dimensiones 3x3 en 2D con un filtro de 2x2 y *stride* de valor 1, extraído de [8]

Los valores que se encuentran en los mapas de características forman combinaciones lineales, por lo que para conseguir encontrar rasgos o propiedades más complejas y no lineales en la imagen se necesitan funciones de activación. Es por ello por lo que después de las capas convolucionales se incluyen siempre estas funciones, siendo la más típica la función ReLU. El resultado de aplicar dichas funciones de activación al mapa de características es un mapa de activación. De esta manera, se obtienen tantos mapas de activación como filtros empleados por cada capa de convolución + ReLU, los cuales se convierten en las entradas de la siguiente capa de la CNN [8].

Para lograr un modelo capaz de detectar características a alto nivel, deben utilizarse diversas capas convolucionales, es decir, la red debe tener profundidad. Cuanto más profunda sea la CNN, mayor capacidad tendrá para detectar patrones más complejos. Esto ocurre ya que, al realizar convoluciones de anteriores convoluciones, la información obtenida aumenta, de manera que en un modelo los filtros de las primeras capas obtienen características de más bajo nivel como bordes o esquinas, mientras que según se avanza en la red la información obtenida es de nivel más alto, como patrones complejos.

Al realizar las convoluciones, el tamaño de la imagen se reduce. Para que esto no ocurra, o si se necesita obtener un tamaño del mapa de activación específico, se introducen bordes de relleno, también conocido como *padding*, en la imagen de entrada, típicamente con valor cero. Al tener valor de cero, estos bordes no afectan al resultado de la convolución, pero sí que permiten obtener el tamaño deseado del mapa de activación. En la Figura 9 se aprecia un ejemplo de *padding* con una imagen de entrada de tamaño 4x4 píxeles.

Figura 9: Ejemplo de *padding* en una imagen 2D con tamaño original de 4x4, extraído de [8]

2.3.1.2. Capas de agrupación (*pooling layers*)

Tras las capas de convolución se encuentran las capas de agrupación, conocidas como *pooling layers*. Estas se encargan de agrupar los valores de salida provenientes de la capa anterior utilizando filtros, típicamente de tamaño 2x2. Esto se realiza ya que con ello se puede conseguir cierto nivel de invariancia ante la rotación y ante las transformaciones geométricas, al reducir la resolución de los mapas de activación [8]. También ayudan a prevenir el sobreajuste al eliminar valores de las capas, y a mejorar la representatividad de las características latentes que detecta la CNN, entre otros beneficios.

Existen diferentes tipos de *pooling*, aunque los más comunes son el *max pooling* y el *average pooling*. En estos, el filtro que es pasado por la imagen ofrece como salida el valor más alto de los píxeles que abarca, o el calcula el valor intermedio entre los píxeles, respectivamente. Se ofrece en la Figura 10 un ejemplo de *max pooling* con una imagen de entrada de tamaño 4x4 píxeles, utilizando un filtro de tamaño 2x2.

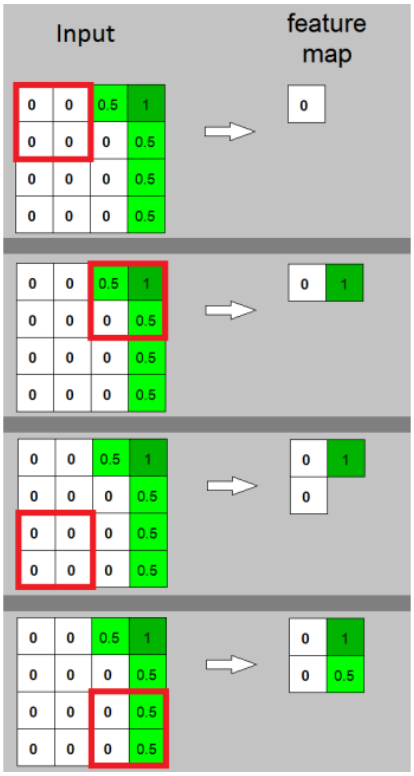

Figura 10: Ejemplo de *max pooling* utilizando un filtro de 2x2, con un *stride* de valor 2, extraído de [8]

Una de las formas más típicas de estructurar este tipo de capas supone usar varias capas convolucionales con su función de activación ReLU, y tras ellas colocar una capa de *pooling*. Esto se

repite en varias ocasiones, usando cada vez una mayor cantidad de filtros en las capas convolucionales, quedando finalmente imágenes de tamaño mucho más reducido que la imagen de entrada. Estas tienen la información y características más destacadas de la imagen a un alto nivel, habiéndose deshecho de los datos innecesarios.

2.3.2. Etapa de clasificación

En esta etapa, los datos y características latentes obtenidas a través de las capas de la etapa anterior se clasifican para obtener una salida final. Típicamente esto se realiza a través de capas completamente conectadas (*fully connected layers*), pero existen también otro tipo de arquitecturas, como los *autoencoders*, de los que se hablará posteriormente.

En las capas completamente conectadas en primer lugar se convierte la última capa proveniente de la etapa anterior en un vector, de manera que se aplanan los datos obtenidos para que solo tengan una dimensión. Tras ello, se introducen estos datos en capas de neuronas completamente conectadas, para procesar las características obtenidas y conseguir finalmente una salida. Para tareas de clasificación la salida final consiste en una etiqueta entre varias disponibles con la clase obtenida según el procesado de la imagen, habiendo sido clasificada según la probabilidad calculada para cada una de las etiquetas posibles según los atributos estudiados por el modelo. Estas suelen utilizar funciones de activación finales como la función Softmax, para llevar a cabo la clasificación. Se muestra un ejemplo completo de un modelo pequeño y resumido en la Figura 11, donde la salida final indica la probabilidad de que la imagen pertenezca a cada una de las tres clases posibles.

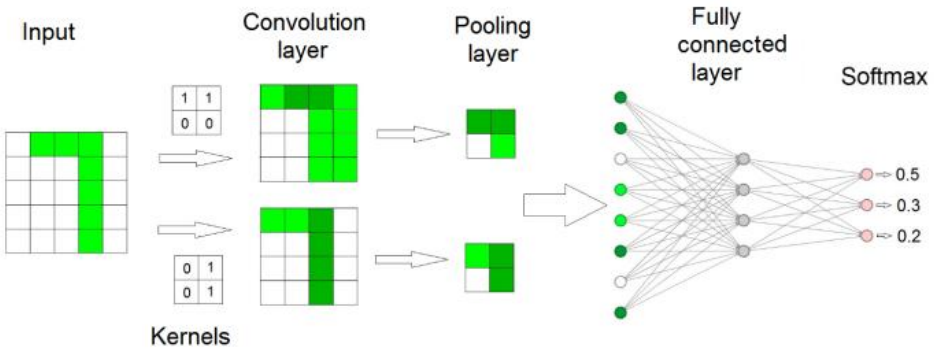

Figura 11: Ejemplo resumido de una CNN, extraído de [8]

2.3.3. Entrenamiento

En las CNN, el entrenamiento tiene el mismo proceso que en las redes neuronales, donde un método de optimización, por ejemplo el descenso de gradiente, debe encontrar los pesos con los que se consigue un menor error en la predicción, calculando este a través de las funciones de pérdida. Sin embargo, en este tipo de redes los pesos se encuentran en los filtros de las capas de convolución, por lo que serán estos los que se actualicen con cada iteración.

2.4. Autoencoders convolucionales

Los autoencoders convolucionales o CAE (*Convolutional Autoencoder*) son un tipo de arquitecturas de redes conocidas como redes completamente convolucionales (*fully convolutional networks*). Como su nombre indica, estos consisten en utilizar la estructura de las capas convolucionales de una CNN, concretamente las de la etapa de extracción de características, sin utilizar capas *fully connected*

pertenecientes a la etapa de clasificación. Esto se debe a que la finalidad de los CAE no es la obtención de etiquetas de una clase a la salida, sino la creación de una nueva imagen de salida a partir de una imagen de entrada. Normalmente se busca conseguir a la salida una reproducción de la imagen de entrada con algunas modificaciones. Algunos ejemplos de las aplicaciones en las que los autoencoders convolucionales destacan en comparación con los resultados obtenidos a partir de otros tipos de autoencoders son la eliminación del ruido en imágenes, y la compresión de imágenes [9][10].

Para realizar este tipo de tarea, la estructura de un CAE se basa en utilizar la parte convolucional de una CNN como codificador, que se asimila a un embudo debido a la reducción progresiva del tamaño de las capas convolucionales debido al uso de las capas de *pooling*, para conseguir así las características latentes de la imagen de entrada, conocidas como representación latente. Esta representación latente se obtiene en lugar de la estructura conocida como cuello de botella o *bottleneck*, el cual es el centro del modelo. Tras ello, se reconstruye la imagen original a partir de la representación latente usando un decodificador, cuya estructura es simétrica al codificador, de manera que se vuelven a utilizar capas convolucionales y la salida final tiene el mismo tamaño de la entrada. En la Figura 12 se ofrecen la estructura general de un CAE con sus tres etapas: codificador, cuello de botella y decodificador, y un ejemplo de un modelo utilizado para la eliminación de ruido, donde se aprecia la estructura general mencionada.

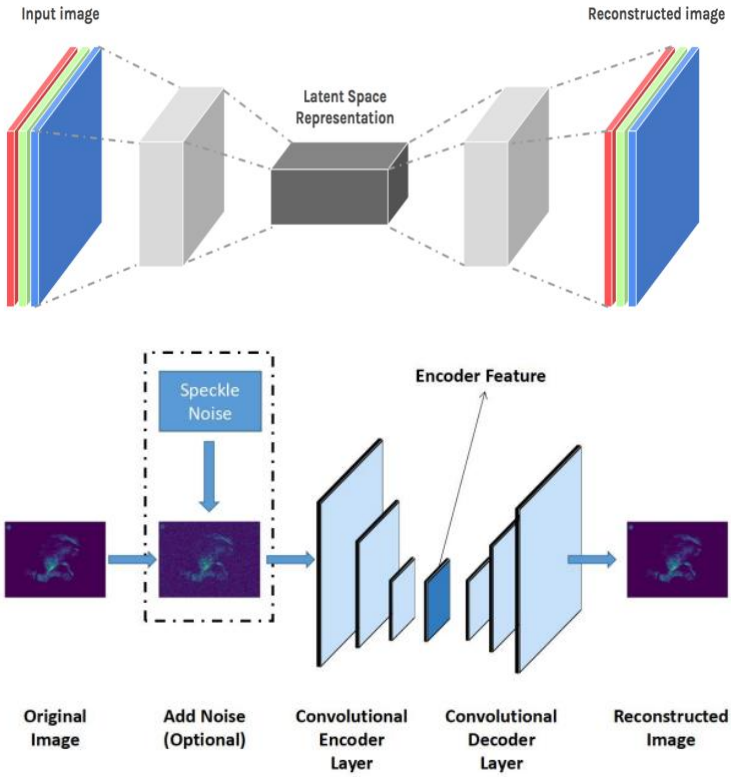

Figura 12: Arriba: Estructura general de un CAE, extraído de [11]. Abajo: Ejemplo de un CAE utilizado para la eliminación de ruido, extraído de [10]

Existen también ciertos modelos de CAE que presentan capas *fully connected*, en especial antes del cuello de botella, pero estas no son el objetivo de estudio de este proyecto.

2.4.1. U-Net

Las U-Net son un tipo de CAE que reciben su nombre debido a la forma de 'U' de su estructura. Esto se debe a que estos autoencoders incluyen conexiones entre capas simétricas del codificador y el decodificador llamadas conexiones de salto, más conocidas como *skip connections*. Los datos de información espacial son complicados de mantener al obtener la representación latente de la imagen, por lo que se usan estas conexiones para poder enviar directamente información espacial desde las capas simétricas al decodificador. De esta manera, en las capas del codificador se utilizan tanto los datos de alto nivel provenientes del *bottleneck* como los datos de menor nivel provenientes de *las skip connections* para la reconstrucción de la imagen. En la Figura 13 se puede apreciar la estructura de una U-net, con las *skip connections* en color gris y su forma en 'U'. Además, se observa en este ejemplo como utiliza capas convolucionales cada vez con más filtros mientras combina estas con las capas de *pooling*, y en la salida combina los datos de alto y bajo nivel.

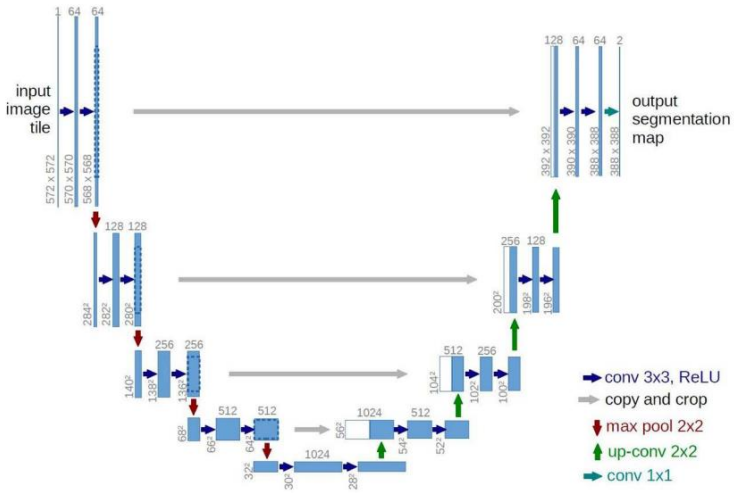

Figura 13: Estructura de una U-Net, extraído de [12]

2.4.2. Autoencoders contractivos

Un autoencoder contractivo es una variante de los autoencoders que incluye un nuevo término de regularización en su función de pérdida, ya que busca que la representación latente encontrada sea menos sensible a variaciones pequeñas en los datos de entrada. De esta manera, su objetivo es que el modelo sea invariante a transformaciones poco importantes, y que así solo aprenda las transformaciones realmente significativas. Esto es importante ya que se consigue así una red más robusta y menos propensa a problemas como el *overfitting*.

La función de pérdida de este autoencoder tendría dos términos:

- La pérdida de reconstrucción, que penaliza el error entre la imagen original y la imagen reconstruida
- La regularización contractiva, que penaliza la norma de Frobenius del Jacobiano de las activaciones ocultas con respecto a la entrada

De esta manera, la pérdida contractiva quedaría tal y como se expone en (5), donde el primer término corresponde a la pérdida de reconstrucción, y el segundo a la regularización contractiva elevada al cuadrado.

$$L = |x - g(f(x))| + \lambda \|J_f(x)\|_F^2 \tag{5}$$

El término de la regularización contractiva se desarrolla según (6), donde al tratarse de la matriz jacobiana, cuyos elementos se definen según (7), se desarrollan las derivadas parciales de cada componente de $h(x)$ respecto a cada componente de x, siendo $h(x)$ las activaciones ocultas y x la entrada. La norma de Frobenius de una matriz se define en (8), y consiste en la raíz del sumatorio de sus elementos al cuadrado. Al elevar todo el término al cuadrado, queda la expresión que se observa en (6).

$$\|J_f(x)\|_F^2 = \sum_{ij} \left(\frac{\partial h_j(x)}{\partial x_i} \right)^2 \tag{6}$$

$$J_{ij}(x) = \frac{\partial h_j(x)}{\partial x_i} \tag{7}$$

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |A_{ij}|^2} \tag{8}$$

De esta manera, si se utiliza este tipo de función de pérdida en un autoencoder este adoptará dicha capacidad de penalización a la sensibilidad, y pequeñas variaciones en la entrada supondrán pequeñas variaciones en la representación.

2.5. Métricas

Para poder valorar la precisión y el desarrollo del entrenamiento del modelo, no solo se utiliza la función de pérdida a lo largo de las épocas, sino que también se utilizan métricas. Las métricas se van calculando a lo largo del entrenamiento, según este avanza a través de los *batches* de cada época. Estas no afectan al resultado del entrenamiento, tal y como ocurre con la función de pérdida, sino que son meramente informativas para el usuario. Se pueden utilizar como métricas cualquier tipo de funciones, aunque dependiendo del modelo a desarrollar hay determinadas métricas las cuales son más adecuadas que otras. En el caso de métricas existentes para medir la calidad entre imágenes se aprecian métricas objetivas y subjetivas. Estas dependen de si se tienen en cuenta solo los resultados numéricos como los valores de los píxeles, o si están basadas en simular resultados obtenidos a través del razonamiento del Sistema Visual Humano (SVH), respectivamente. Algunas de las métricas más extendidas para el cálculo de la calidad de las imágenes son el PSNR, SSIM y SSIM multiescala (MS-SSIM).

2.5.1. PSNR

El PSNR (*Peak Signal-to-Noise Ratio*) entre dos imágenes, siendo f la imagen de referencia y g la imagen predicha, ambas del mismo tamaño *MxN* se define según la expresión (9). En ella el MSE se calcula según la expresión (2) vista anteriormente, donde se calcula la diferencia entre cada uno de los píxeles de ambas imágenes.

$$PSNR(f, g) = 10 \log \left(\frac{255}{MSE(f, g)} \right) \tag{9}$$

De esta manera, cuanto mayor sea el valor del PSNR (medido en dB), menores serán las diferencias entre ambas imágenes numéricamente, suponiendo así una mayor calidad de la imagen. Esta métrica ha demostrado buenos resultados para evaluar la calidad de imágenes con ruido. Sin embargo, diversos estudios califican que el PSNR puede tener fallos cuando hay cambios en la estructura de la imagen, ya que diversos cambios estructurales pueden ofrecer el mismo valor de MSE mientras que para el SVH habrá cambios que supongan más diferencias que otros [13].

2.5.2. SSIM

El SSIM (Structural Similarity Index Measure), es una métrica que permite medir la similitud entre dos imágenes, y se considera que tiene correlación con la percepción de la calidad según el SVH [14]. Este está diseñado de manera que se tienen en cuenta para su cálculo tres términos diferentes: la luminancia, el contraste y la estructura, según la expresión (10). La luminancia $l(f,g)$ sigue la expresión (11), donde se compara el brillo medio de ambas imágenes (μ_f, μ_g) para comprobar su cercanía. El contraste $c(f,g)$ sigue la expresión (12), el cual compara y mide la cercanía del contraste de ambas imágenes (σ_f, σ_g). Este se mide a partir de la desviación estándar. La estructura $s(f,g)$ se mide a través de la expresión (13), que analiza el coeficiente de correlación entre ambas imágenes, siendo σ_{fg} la covarianza entre f y g. Se usan además las constantes C_1, C_2 y C_3 para evitar denominadores nulos.

$$SSIM(f,g) = l(f,g)\, c(f,g)\, s(f,g) \tag{10}$$

$$l(f,g) = \frac{2\mu_f\mu_g + C_1}{{\mu_f}^2 + {\mu_g}^2 + C_1} \tag{11}$$

$$c(f,g) = \frac{2\sigma_f\sigma_g + C_2}{{\sigma_f}^2 + {\sigma_g}^2 + C_2} \tag{12}$$

$$s(f,g) = \frac{\sigma_{fg} + C_3}{\sigma_f\sigma_g + C_3} \tag{13}$$

De esta manera, el SSIM resultante queda en una escala entre 0 y 1, donde un valor de 0 indica una correlación nula por lo que las imágenes tienen estructuras muy diferentes, y 1 indica que $f = g$. Como el SSIM tiene en cuenta el contexto de la imagen, su resultado se acerca más a la percepción humana.

2.5.3. MS-SSIM

El SSIM multiescala o MS-SSIM es una variante de la métrica anterior, en la cual se calcula el valor del SSIM en múltiples escalas espaciales de la imagen. Estas escalas se calculan a través de un proceso de filtrado y submuestreo, creando así una pirámide de imágenes para la imagen de referencia y otra para la imagen de prueba. Cada nivel de la pirámide supone una escala diferente, siendo todas ellas versiones reducidas de la imagen original. Tras calcular el SSIM en cada escala, se aplica una ponderación a cada uno de los resultados en función de la importancia relativa de cada escala respecto a la percepción de la calidad en la imagen. Finalmente, se combinan los resultados de todas las escalas para obtener el valor del MS-SSIM tal y como se muestra en la expresión (14), donde M es el número de escalas, y α_j, β_j y γ_j son los pesos asignados a la luminancia, contraste y estructura en cada escala.

$$MS - SSIM(x,y) = \prod_{j=1}^{M} [l_j(x,y)]^{\alpha_j} \cdot [c_j(x,y)]^{\beta_j} \cdot [s_j(x,y)]^{\gamma_j} \quad (14)$$

La división en escalas para el cálculo del MS-SSIM se debe a que ciertos tipos de distorsiones no son igual de perceptibles en cualquier tipo de escala, sino que algunas pueden detectarse mejor en una escala de detalle fino, mientras que otras pueden ser vistas en escalas más gruesas o a nivel global. Este razonamiento proviene del SVH, ya que la percepción humana no es uniforme a través de todas las frecuencias espaciales.

2.6. Trabajos previos y estado del arte

En el ámbito en el que se encuentra este proyecto se hallan otros trabajos previos, de naturaleza más compleja. En el pasado ha habido numerosos modelos para arreglar errores de subexposición o sobreexposición individualmente, en especial del primero. Algunos ejemplos de estos modelos se observan en artículos como [15] o [16], donde ambos tratan la corrección de imágenes subexpuestas con diferentes tipos de arquitecturas.

Sin embargo, este proyecto busca poder solucionar ambos tipos de error de exposición. Esto ha sido tratado más recientemente, con otro tipo de arquitecturas. El principal modelo que ha propuesto una solución es el publicado por Afifi et al. en [2]. Este consigue que al introducir una imagen con un error de exposición en el modelo se devuelva dicha imagen corregida. Para ello utiliza un complejo sistema que incluye 3 U-Nets, donde se trata un trozo recortado de la imagen de entrada, descomponiéndola en diferentes capas utilizando una pirámide laplaciana. Se muestra la estructura de dicho sistema en la Figura 14.

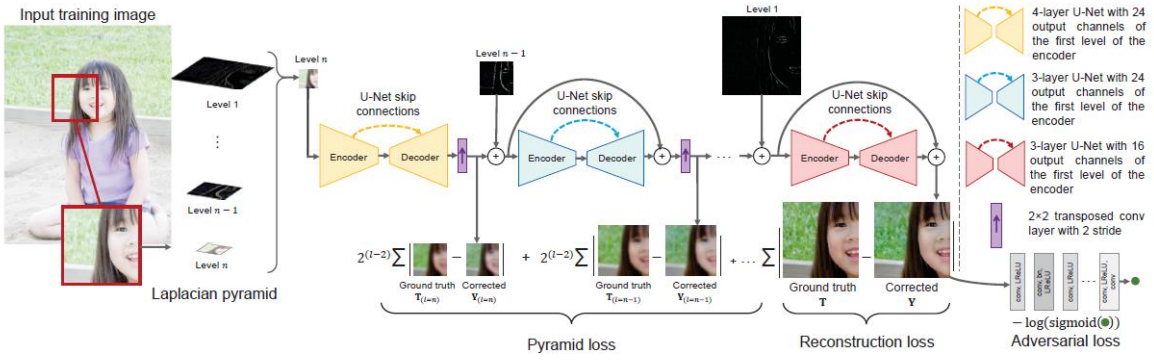

Figura 14: Estructura general del modelo de Afifi et al. [2]

Para entrenar el modelo crea su propia base de datos, accesible en [13]. Los resultados obtenidos con este sistema son muy buenos ya que no solo consigue corregir los errores de exposición, sino que también lleva a cabo un realce de los colores de las imágenes, además de obtener valores muy altos en las métricas estudiadas (PSNR, SSIM) respecto a otros modelos anteriores.

Otro sistema que destaca en la corrección de errores de exposición ha sido el que se ofrece por Eyiokur et al. en [17]. Este utiliza la base de datos creada por Afifi et al., de manera que hay algunas arquitecturas comunes en su propuesta. La estructura del sistema consiste en un modelo de U-Net complejo para generar una imagen a través de la imagen de entrada, a la que se le añaden diversos bloques adicionales a su salida. Estos bloques adicionales y su efecto se incluyen en el cálculo de la función de pérdida, de manera que se consigue con ellos mejorar la calidad de la imagen generada y el comportamiento del modelo.

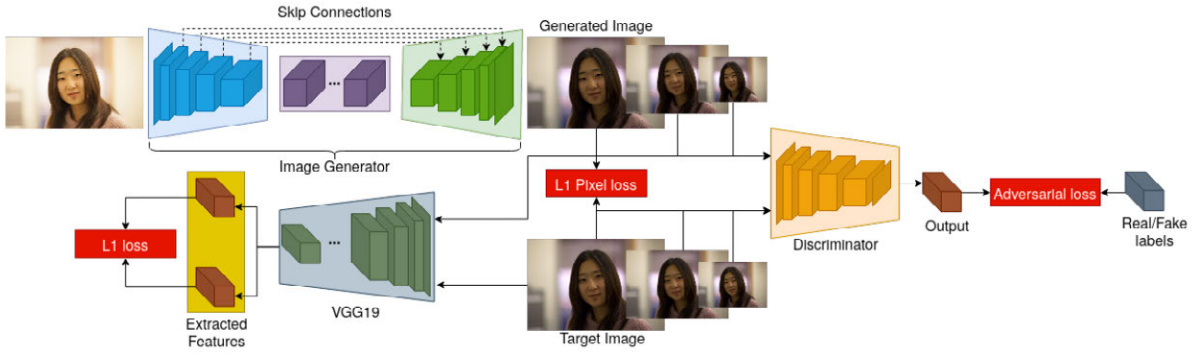

Figura 15: Estructura del sistema propuesto por [17]

Los resultados obtenidos con este sistema se miden con las mismas métricas que el sistema anterior (PSNR, SSIM), y estos se comparan directamente con los resultados de Afifi et al. En todas las categorías estudiadas el modelo [17] obtiene mejores resultados que el anterior. Las imágenes generadas también demuestran visualmente que el modelo tiene un buen funcionamiento.

3. Especificaciones y restricciones de diseño

Para el proyecto existen una lista de especificaciones, requisitos y restricciones, y se presentan estas a continuación.

- Para el diseño del modelo se deben utilizar arquitecturas de autoencoders contractivos, concretamente siguiendo una estructura de modelo tipo U-Net.
- El codificador y el decodificador deben tener una serie de capas diferentes según su posición dentro del modelo, que estén organizadas de manera simétrica a ambos lados del cuello de botella.
- El modelo debe tratar con imágenes en escala de grises, realizando en caso de ser necesaria una conversión desde el espacio RGB.
- El código para desarrollar el modelo se debe escribir en el lenguaje de programación Python, concretamente en su versión Python 3.10 en adelante.
- Para el desarrollo del código se utilizan librerías especializadas en la creación de modelos de aprendizaje automático, como Tensorflow y Keras, junto a otras librerías como Skiimage.io o Matplotlib para la representación y tratamiento de las imágenes.
- El entorno de trabajo a utilizar son cuadernos interactivos tipo Jupyter Notebook, utilizando concretamente el software de Google Colaboratory, en su versión de membresía Colab Pro.
- El modelo utiliza para sus fases de entrenamiento, validación y pruebas una base de datos de imágenes con errores de subexposición y sobreexposición, junto con sus respectivas versiones con dichos errores corregidos que se utilizan como *ground truth*.
- El código final debe tener como entrada una imagen con un error de exposición (subexposición o sobreexposición) y obtener como salida dicha imagen corregida. El modelo se encarga a través de sus capas de corregir el error de exposición sin cambiar los datos u organización original de la fotografía, es decir, sin omitir elementos originales de esta como objetos, personas, u otras entidades presentes.
- El modelo se limita a corregir los errores de exposición, no trata otros errores como la eliminación de ruido ni realiza ningún realce ni restauración de la imagen.
- Al tratarse de un modelo más simple en comparación a otros ya existentes, se buscan obtener los mejores resultados dentro de sus capacidades.
- Es necesario un preprocesado de las imágenes adecuado a las necesidades del modelo, que también depende de la base de datos a utilizar y de las especificaciones de las librerías utilizadas en el código.

4. Descripción de la solución propuesta

El problema que se dispone quiere solucionarse a través de la creación de un modelo que utilice la arquitectura de una U-net convolucional contractiva. De esta manera, se busca crear un sistema más sencillo que los ofrecidos por [2] y [17], pero usando estos dos modelos ya existentes como base para la idea inicial, para comprobar si el proyecto pudiera servir como una alternativa simple. Esta es una de las razones por las que se elige para el entrenamiento el mismo *dataset* que usan ambos modelos, descargado desde [13]. Al tratarse este proyecto de una aproximación más sencilla, las imágenes elegidas para que trate el modelo se encuentran en escala de grises, a diferencia de los modelos mencionados previamente. Esto conlleva un menor coste computacional y un modelo algo menos complicado de tratar y entrenar.

Antes de introducir las imágenes en el modelo, se lleva a cabo un preprocesado para adaptarlas a la arquitectura de la red, con tareas como la conversión de las imágenes desde RGB hasta escala de grises, entre otras. Como se busca que el autoencoder sea contractivo, la función de pérdida a utilizar se basa en la expresión (5), adaptándola al modelo a través de la creación de una función propia. Para medir el progreso del entrenamiento y del desarrollo del modelo se utilizan una serie métricas, siendo estas el MSE, PSNR, SSIM y MS-SSIM. Al finalizar el entrenamiento del modelo se realizan diversas pruebas para comprobar su funcionamiento y eficacia. El proceso general que se lleva a cabo en el proyecto se resume en la Figura 16, donde se aprecia el esquema general del funcionamiento del modelo.

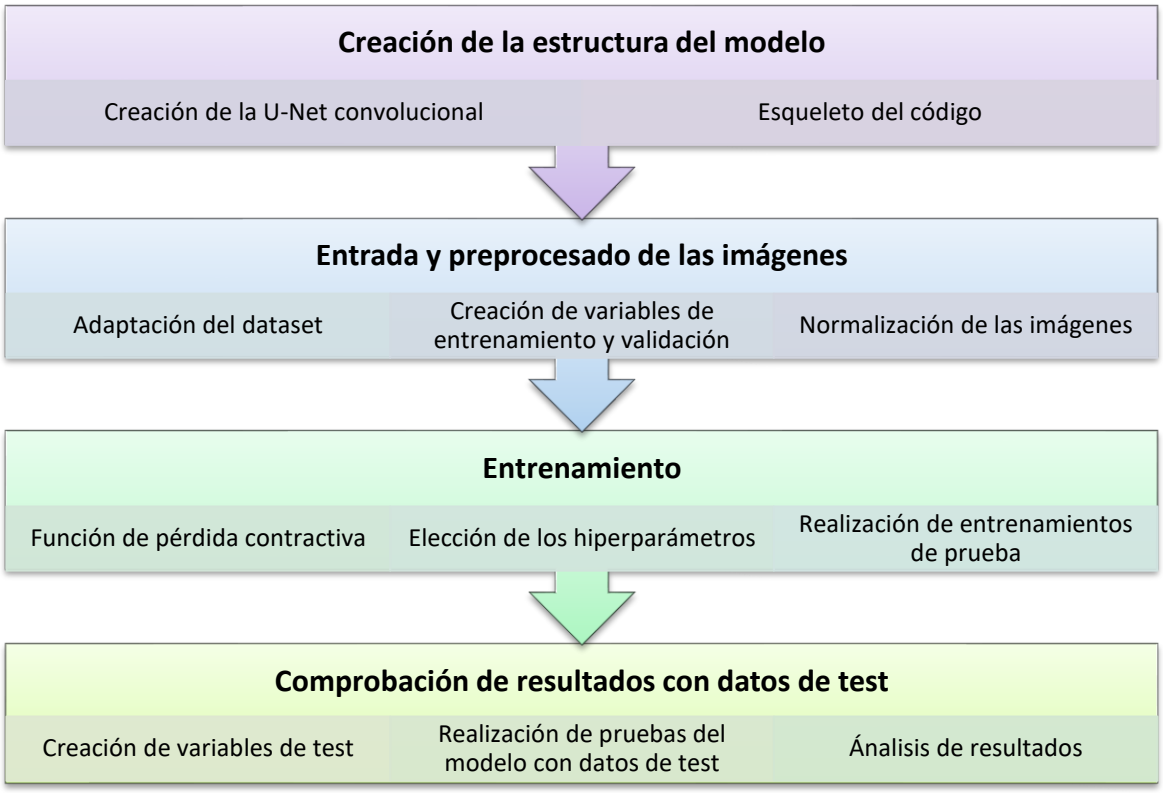

Figura 16: Esquema general del modelo.

Todo el código del modelo se desarrolla en Python, a través de la herramienta Colab Pro, de manera que se utiliza también la herramienta de Google Drive para almacenar todos los ficheros creados, los archivos de imágenes correspondientes a las bases de datos, etc. Para el código se utilizan diversas librerías especializadas en el desarrollo de modelos de aprendizaje automático, en especial Tensorflow

[18] y Keras [19]. Para la lectura y representación de imágenes se utilizan otras librerías como scikit-image [20] y Matplotlib [21]. Otras librerías utilizadas para la gestión de archivos y directorios son os [22] y JSON [23]. Como librería adicional multifuncional se implementa también NumPy [24].

4.1. Estructura

La estructura del modelo sigue la de una U-net, de forma que existen 3 etapas diferenciadas: el codificador, el cuello de botella y el decodificador. El codificador y el decodificador son simétricos, y sus capas internas están unidas a través de *skip connections*.

El modelo final se muestra en la Figura 17, y cuenta con una estructura conformada por 3 niveles de capas convolucionales en el codificador y en el decodificador. Cada uno de estos niveles incluye dos capas convolucionales con activación ReLU, junto a una capa de *max pooling* en el caso del codificador, y una capa de *upsampling* y otra de concatenación en el decodificador. La capa de concatenación es utilizada para la unión de la información latente proveniente del cuello de botella y la información proveniente de las *skip connections*. Para el cuello de botella se implementan tres capas convolucionales. Como capa final del modelo se utiliza una capa convolucional con activación sigmoide, para la reconstrucción de la imagen final. Además, cada uno de los niveles cuenta con capas de *Dropout* al finalizar, estableciendo así una regularización para evitar el *overfitting*.

Todas las capas convolucionales cuentan con filtros de tamaño 3x3 píxeles, donde el número de filtros utilizado depende del nivel de la capa. El nivel superior comienza con 32 filtros, duplicando su valor según se avanza de nivel y llegando al cuello de botella con 256 filtros. Las capas de *max pooling* y de *upsampling* utilizan filtros de tamaño 2x2 píxeles. La capa convolucional final del programa utiliza un único filtro, ya que la imagen predicha se encuentra en escala de grises.

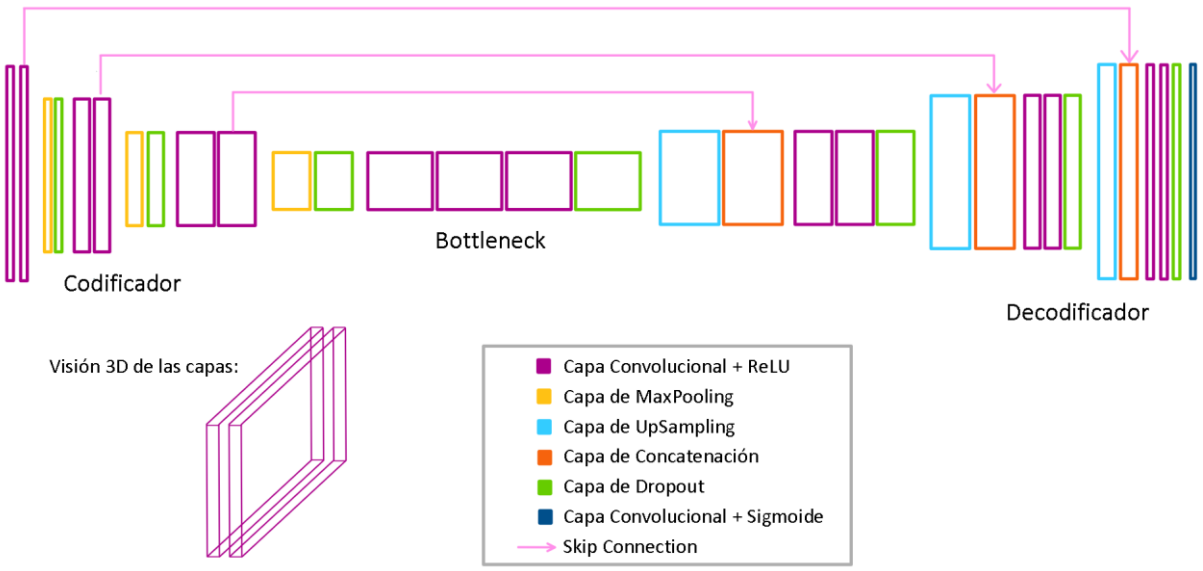

Figura 17: Diagrama de la estructura final del modelo

Todas las capas utilizadas en este modelo utilizan para su implementación funciones de Keras, concretamente de la clase *Layers*. Estas se listan a continuación:

- *Conv2D*[25] para las capas convolucionales
- *MaxPooling2D* [26] para las capas de *max pooling*
- *UpSampling2D* [27] para las capas de *upsampling*

- *Concatenate* [28] para las capas de concatenación
- *Dropout* [29] para las capas de *dropout*

Cabe destacar que no se ha utilizado esta estructura para todas las pruebas, ya que se comienzan los entrenamientos de prueba con una versión algo más sencilla, la cual no incluye las capas de *dropout* ni la tercera capa del *bottleneck*. La introducción de las nuevas capas se explica junto a las pruebas y resultados obtenidos más adelante.

4.2. Entradas

El modelo propuesto tiene como entradas imágenes con errores de subexposición o de sobreexposición, junto a imágenes que no tienen errores de exposición, todas ellas provenientes del *dataset* de [2]. Para introducirlas al modelo, estas deben primero agruparse en *batches*, y cada una de ellas debe tener una imagen de referencia *ground truth* con la que poder calcular el error de la predicción ofrecida por el modelo. Para realizar el entrenamiento, se ofrecen al modelo ambas imágenes enlazadas (entrada y *ground truth*), que deben pasar por un procesado previo para mejorar la eficacia y el correcto funcionamiento del sistema.

4.2.1. Base de datos

La base de datos a utilizar, que fue creada para el sistema de [2], consiste en 5000 imágenes diferentes provenientes de otra base de datos, siendo esta MIT-Adobe FiveK dataset [30]. Este *dataset* [30] consiste en 5000 imágenes coleccionadas gracias a la aportación de diferentes fotógrafos, todas ellas en formato RAW[1] . Este formato muestra las fotografías originales sin retoques. El *dataset* aporta también las mismas fotografías editadas por 5 fotógrafos profesionales de manera individual, de manera que se realizan ligeras correcciones de iluminación y color, entre otras, para mejorar el aspecto general de las imágenes. Para la base de datos que se crea en [2], se utilizan las imágenes originales en formato RAW y las imágenes resultantes creadas por el tercer fotógrafo (referido en [30] como "*Expert C*").

Para simular imágenes con errores de sobreexposición y de subexposición, Afifi et al. [2] utilizan un programa externo para renderizar las imágenes en formato RAW, y aplican 5 valores de EV respecto a su valor de EV original. De esta manera, de cada imagen RAW se renderizan 5 versiones, teniendo estas un valor de EV relativo de -1'5, -1, +0, +1, +1'5. El valor +0 correspondería con el valor de exposición original de la fotografía. En la Figura 18 se muestra un ejemplo del efecto de estas reducciones y ampliaciones del valor de EV en una de las imágenes. Tal y como se puede apreciar, se simula correctamente el efecto de la subexposición y de la sobreexposición.

[1] El formato RAW en fotografía es un tipo de archivo que genera el sensor de la cámara y contiene toda la información y datos de la imagen capturados, pero sin estar procesados ni comprimidos, a diferencia de otros formatos como JPEG. En ellos se mantiene la máxima cantidad de información posible con la mayor calidad, permitiendo así una mayor flexibilidad para un post-procesado. Por ello, es el formato elegido siempre por parte de fotógrafos profesionales para guardar las fotografías tomadas.

Figura 18: Ejemplos de imágenes del *dataset* [2]. De izquierda a derecha, con valores de EV relativo respecto a su original de -1'5, -1, +0, +1, +1'5

De esta manera, de las 5000 imágenes iniciales se obtienen 5 versiones de cada una para utilizar como imágenes de entrada, consiguiendo así 25000 imágenes. Sin embargo, se reduce el número total a 24330 imágenes, al hallar problemas con ciertas imágenes y su renderizado.

Para las imágenes de *ground truth* se utilizan las imágenes retocadas profesionalmente por "*Expert C*" [30]. Afifi et al. utilizan estas imágenes en vez de las imágenes de entrada con valor de exposición relativo de +0, ya que muchas de las fotografías originales presentaban errores de exposición debido a fenómenos de iluminación como el contraluz. Se presenta un ejemplo de esto en la Figura 19, donde "*Expert C*" ha corregido el error que presentaba la imagen original.

Figura 19: Izquierda: Imagen de entrada con valor de EV relativo de +0. Derecha: Imagen *ground truth*, retocada por "*Expert C*" [30]

De esta manera, quedarían 24330 imágenes de entrada, con sus respectivas 4866 imágenes *ground truth*. Estas se dividen en tres grupos para las diferentes etapas del modelo, quedando estas como se presenta a continuación:

- Etapa de entrenamiento (*training*): 17675 imágenes de entrada, con 3535 imágenes *ground truth*
- Etapa de validación (*validation*): 750 imágenes de entrada, con 150 imágenes *ground truth*
- Etapa de pruebas (*testing*): 5905 imágenes de entrada, con 1181 imágenes *ground truth*

Para la etapa de pruebas Afifi et al. utilizan imágenes no solo de "*Expert C*", sino también de los otros editores. En este proyecto solo se han contemplado como imágenes *ground* truth las corregidas por "*Expert C*".

Todas las imágenes pertenecientes al *dataset* se guardan en un árbol de directorios en Google Drive, en función de la etapa a la que correspondan (entrenamiento, validación o pruebas). En cada uno de estos directorios también se diferencian entre las imágenes de entrada y las de *ground truth*. Se muestra la organización de estos directorios en la Figura 20.

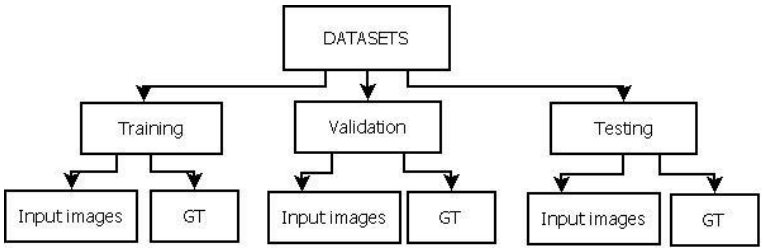

Figura 20: Árbol de directorios con la organización de las imágenes del *dataset*

4.2.2. Preprocesado e introducción de las imágenes en el modelo

Las imágenes deben introducirse al modelo en pareja, donde cada entrada debe tener asignada su *ground truth*. Como hay 5 versiones de cada imagen que comparten la salida, se crea una función auxiliar que se encarga de quintuplicar las imágenes de *ground truth* (GT), de manera que haya una salida por cada entrada. Esta lee las imágenes de GT del directorio correspondiente (ver Figura 20) y copia su nombre original, para guardar luego 5 copias de esta imagen en un nuevo directorio con el formato de nombre de fichero de: (nombre original de la imagen_numero de copia.jpg). Por ejemplo, la copia número 3 de la imagen *a0001-jmac_DSC1459.jpg* se guarda como: *a0001-jmac_DSC1459_3.jpg.*

Se proporciona el código de la función mencionada en el Anexo A.1. Cabe destacar que esta función solo se utiliza una vez al inicio cuando se quieren adaptar las salidas de los *datasets* de entrenamiento, validación o pruebas.

Además, todas las imágenes, tanto entradas como *ground truth*, deben someterse a un preprocesado digital antes de ser introducidas al modelo. Para ello se crea otra función auxiliar, donde se realiza la normalización de los datos utilizando la función de la biblioteca Keras *Rescaling* [31]. Esta función se nombra como *preprocesar_imagenes*.

Para introducir las imágenes del *dataset* en una variable y poder trabajar con ellas, se utiliza la función de Tensorflow *image_dataset_from_directory* [32], que permite crear variables tipo *Dataset* (pertenecientes a Tensorflow, se pueden ver sus especificaciones en [33]) a partir de las imágenes en un directorio. Se crean de esta manera diferentes variables para contener las imágenes de entrada y GT de las tres etapas, quedando así 6 variables según los directorios que aparecen en la Figura 20. Para que las imágenes de entrada y sus respectivas GT sigan el mismo orden de almacenamiento al estar en dos variables separadas, se establecen diversos parámetros en la función mencionada en [32], entre ellos la desactivación de la aleatorización del orden. Otro de los parámetros importantes a mencionar es *color_mode*, con el que se transforman las imágenes a escala de grises, al ser estas el tipo de imágenes con las que trabaja el modelo.

Tras crear las 6 variables tipo Dataset, se utiliza la función creada anteriormente *preprocesar_imagenes* para la normalización de estas. Al tratarse de una función externa, se aplica a las variables a través de la función de Tensorflow *map*, presente en las especificaciones [33]. Finalmente, para unir las variables según su etapa de manera que se enlacen en una única variable las imágenes de entrada y salida de entrenamiento, validación y pruebas, se utiliza la función de Tensorflow *zip*, también disponible en [33]. De esta forma, se tienen tres variables tipo Dataset en las que se encuentran las imágenes enlazadas de entrada y GT. Este proceso se presenta en forma de esquema en la Figura 21.

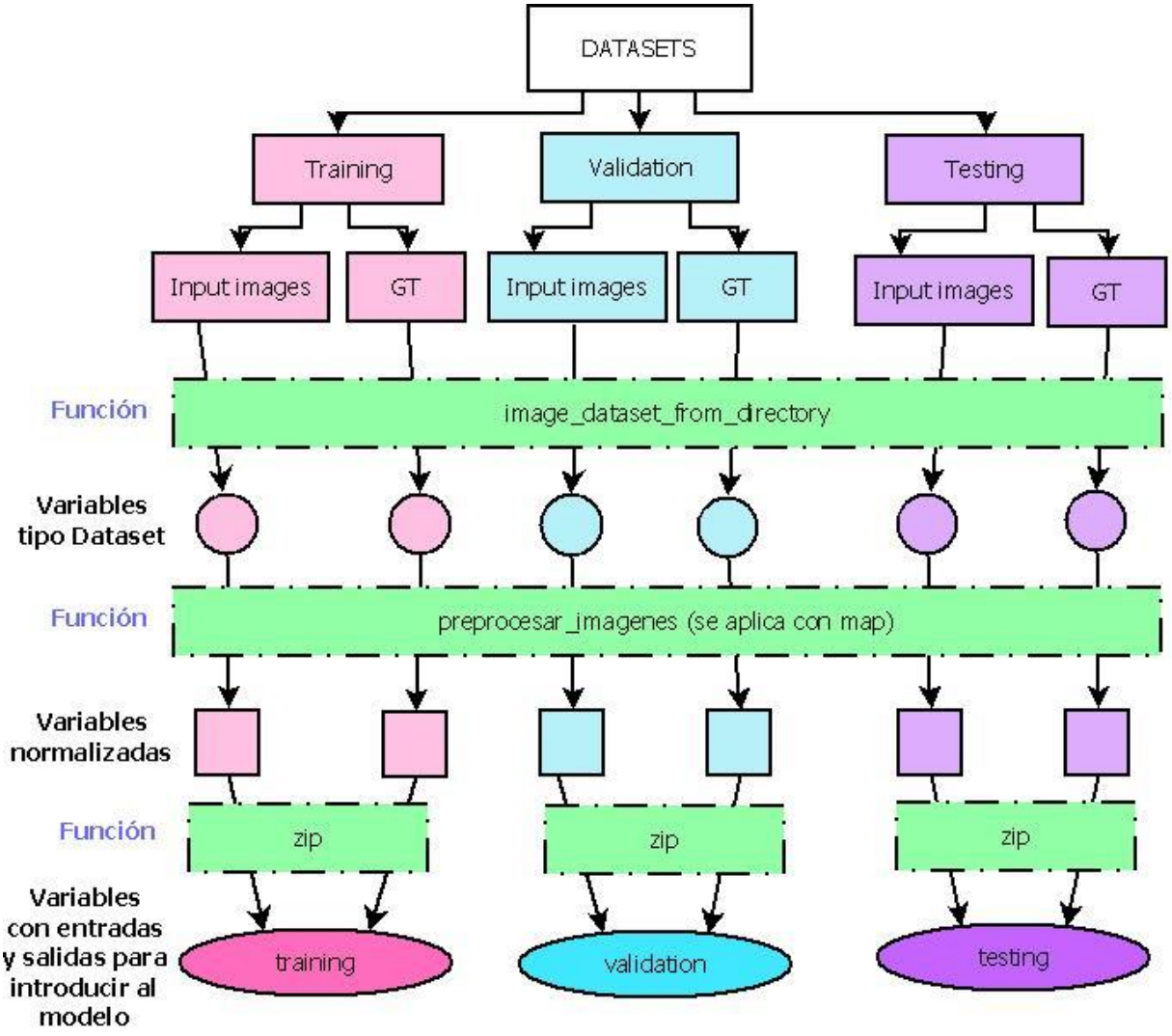

Figura 21: Esquema del proceso que siguen las imágenes de entrada

Además, otro parámetro que se elige al utilizar *image_dataset_from_directory* es el tamaño del grupo (*batch size*) que se quiere utilizar, quedando así las imágenes distribuidas en batches dentro de la variable. De esta manera, las tres variables finales que contienen las imágenes de cada etapa se asimilan en estructura similar a una tupla con las entradas y salidas asociadas. La estructura de estas variables se muestra de manera más detallada en la Figura 22.

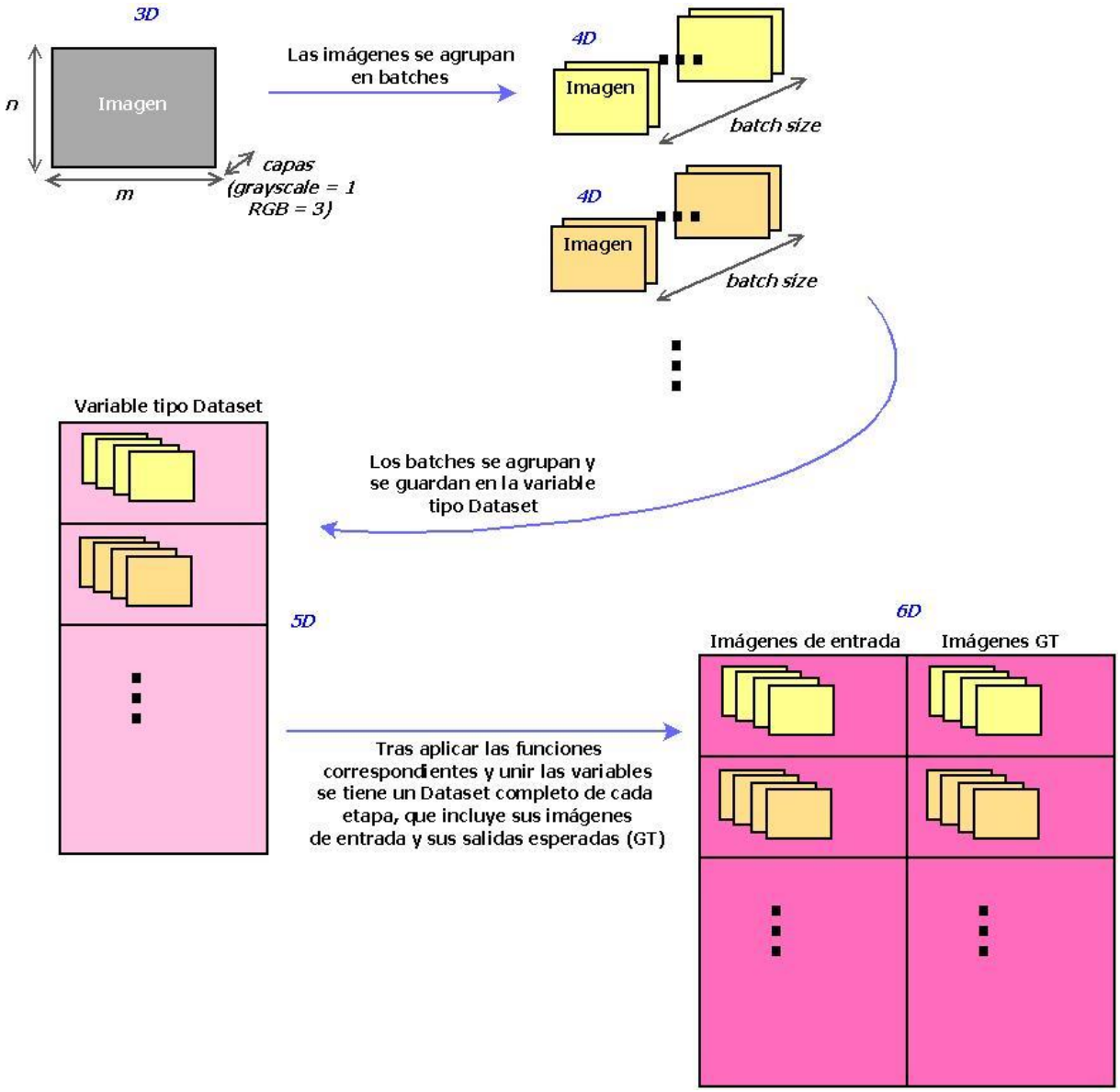

Figura 22: Esquema de la organización de las imágenes

4.3. Entrenamiento

Para la fase de entrenamiento el modelo se configura con una función de pérdida para calcular los pesos a través de la retropropagación y de un optimizador. El optimizador elegido ha sido *Adam*[34], el cual es utilizado comúnmente en modelos de aprendizaje profundo debido a su eficiencia y efectividad. Esto se debe a que *Adam* se encarga de realizar el descenso de la magnitud de los gradientes ajustando las tasas de aprendizaje (*learning rates*) individualmente para cada parámetro, en vez de utilizar un *learning rate* constante para todos los parámetros.

El entrenamiento hace uso de las variables de *training* y *validation*. Esto ocurre ya que el modelo utiliza los *batches* de imágenes de *training* en cada época para configurar los pesos, y al finalizar realiza una ronda de prueba con imágenes diferentes que no conoce, siendo estas las de *validation*, para poder realizar un entrenamiento correcto.

De esta manera, la fase de entrenamiento comienza con creación de una función de pérdida contractiva. Tras ello se lleva a cabo la definición de los hiperparámetros y la inicialización del

optimizador *Adam* a través de la función de Tensoflow *Adam*[34]. También se configuran las métricas a medir. Con todos los parámetros y funciones ya definidas se utilizan las funciones que Keras ofrece para el entrenamiento de un modelo, siendo en primer lugar la compilación del modelo con *compile*, y tras ello la realización del entrenamiento con *fit*. Ambas funciones cuentan con diversos argumentos e indicaciones[35]. Este proceso se muestra en la Figura 23.

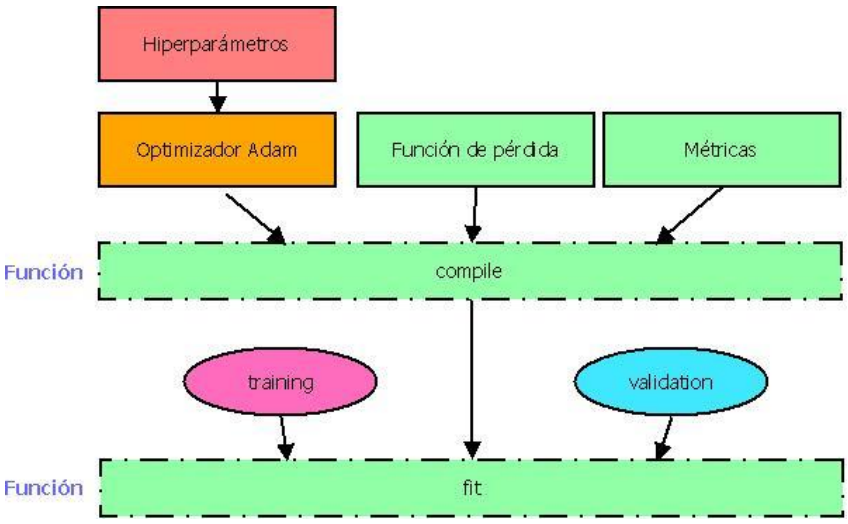

Figura 23: Esquema del entrenamiento del modelo

Para poder encontrar el valor de los hiperparámetros y argumentos más adecuados con los que alcanzar la mayor eficacia posible del modelo se llevan a cabo una larga serie de entrenamientos. Sus resultados y conclusiones se analizan más adelante en el apartado de Resultados.

4.3.1. Función de pérdida

La función de pérdida que utiliza la red es una función de pérdida contractiva, y se codifica desde cero definiendo una nueva función. La base de esta es la expresión (5), de manera que la función se ocupa de calcular en primer lugar el MSE, y a esto se suma la parte contractiva.

El MSE se puede calcular a través de la función de Keras *MeanSquaredError*[36], mientras que la parte contractiva necesita obtener los pesos de las capas del modelo. Para poder acceder a los pesos y hacer cálculos con los gradientes se utilizan funciones de Tensorflow, mientras que las capas del modelo pertenecen a Keras (clase *Layers*), por lo que para poder utilizar estas funciones sobre las capas de Keras se crea una nueva clase que hereda de la clase *Layers* de Keras[37]. Esta nueva clase recibe el nombre de *ContractiveLossLayer*, y se crean en ella un constructor y la función de pérdida contractiva que utiliza el modelo. Se puede consultar la clase en el Anexo A.2.

El único argumento con el que cuenta la clase en su inicialización es el valor de lambda, siendo la constante que se aprecia en la expresión (5). Al llamar a la función se introducen también la imagen predicha por el modelo (imagen de salida) y la imagen de *ground truth*, para realizar las operaciones entre ambas.

En primer lugar, la función calcula el MSE. Tras ello, el cálculo de la parte contractiva comienza con el cálculo de los gradientes. Para ello se utiliza la imagen predicha anteriormente y se introduce esta como entrada del modelo, lo que sirve para medir cómo de sensible es la reconstrucción a pequeños cambios en la representación latente. Esto se debe a que como la imagen predicha es una imagen que ha sido creada por el modelo a partir de la representación latente de la imagen de entrada, si esta se

introduce como entrada después la representación latente es muy similar a la anterior, al provenir de otra representación latente y siendo los cambios menores entre la imagen *ground truth* y la reconstrucción. De esta manera se puede medir cómo pequeños cambios en la representación latente afectan a la reconstrucción final, pudiendo penalizarlos para que el modelo sea menos sensible a pequeños cambios, de forma que dos representaciones latentes similares consigan una reconstrucción parecida. Para ello se calculan los gradientes entre la imagen predicha y la reconstrucción creada a partir de ella, a través de la estructura de Tensorflow *GradientTape*[38]. Después del cálculo de los gradientes, se calcula la norma de Frobenius con todos sus componentes y se multiplica el sumatorio del resultado por lambda, teniendo así el término contractivo de la expresión(5). Finalmente, la función retorna la suma del MSE y la parte contractiva.

Para utilizar esta pérdida en el modelo, se incluye al utilizar *compile*, concretamente en el atributo *loss.*

4.3.2. Hiperparámetros

Los hiperparámetros que se deben tener en cuenta en el funcionamiento del modelo son el número de épocas de entrenamiento, el *batch size*, el valor inicial del *learning rate* y el valor de lambda en la función de pérdida. Para elegir estos se realizan una larga lista de entrenamientos de prueba, explicados en el apartado de Resultados.

Estos son importantes para regular problemas como el *overfitting* o el *underfitting*. En esto se puede ver el efecto del número de épocas, el cual debe ser lo suficientemente alto para que el modelo quede bien entrenado, pero sin que haya un sobreajuste. Se contemplan para las pruebas diferentes cantidades de épocas, entre 6 y 20. Este parámetro se incluye al utilizar la función *fit*.

También afecta directamente al *overfitting* el *batch size*, ya que el modelo debe tener las suficientes imágenes en cada grupo como para que no se ajuste a sus características en exceso, mientras que el modelo no debe sobrepasar la capacidad de memoria del programa. Para las pruebas se mantiene el valor de 32. En el modelo se escoge el *batch size* a la hora de introducir las imágenes en el programa con *image_dataset_from_directory*.

Un *learning rate* inicial alto puede llevar a que el modelo no encuentre el mínimo de la pérdida al converger demasiado rápido, pero un *learning rate* inicial demasiado bajo puede llevar a que la pérdida se estanque. En las pruebas se utilizan valores iniciales entre 10^{-4} y 10^{-2}. Este valor se introduce al inicializar el optimizador con la función *Adam*.

Lambda es un hiperparámetro que regula el equilibrio entre la minimización del error de reconstrucción y la penalización de la sensibilidad del modelo a pequeños cambios en las entradas. Un valor pequeño de lambda implica que la sensibilidad a los pequeños cambios es mayor, mientras que un valor alto de lambda crea un modelo robusto con una regularización fuerte, menos sensible a los cambios. Los principales valores de lamda utilizados en las pruebas han sido de 0,1 y de 0,01. Este valor se introduce al inicializar la función.

4.3.3. Métricas

Las métricas que utiliza el modelo para medir los resultados en cada una de las épocas son MSE, PSNR, SSIM y MS-SSIM. Estas se incluyen en la función *compile*. Existen funciones de Tensorflow para el cálculo del PSNR, SSIM y MS-SSIM, todas ellas pertenecientes a la clase *tf.image,* pero como *compile* pertenece a Keras por compatibilidad se definen tres funciones nuevas que utilizan dentro de ellas

cada una de las funciones de Tensorflow. Para el cálculo del MSE se utiliza una de las funciones preexistentes de Keras directamente, de la clase *keras.metrics*.

4.4. Etapa de pruebas

Tras el entrenamiento del modelo, se realizan pruebas de este a través del uso de imágenes no conocidas, siendo estas las imágenes de *testing*. Para ello, Keras ofrece las funciones *evaluate* y *predict* [35], las cuales se encargan de realizar las pruebas con los datos proporcionados y generar predicciones en función de las imágenes de entrada, respectivamente.

Tras ello, se desnormalizan las imágenes predichas y se comprueban visualmente los resultados.

5. Resultados

La comprobación de resultados del modelo supone la realización de una larga serie de entrenamientos de prueba, en los cuales se analiza el efecto de los hiperparámetros para encontrar los valores ideales. Si se obtienen buenos resultados en estos entrenamientos, se procede a evaluar el modelo y a generar predicciones con los datos de prueba. Durante todas las pruebas se comprueba también la necesidad de añadir nuevas opciones o capas en el modelo, según los resultados que se obtienen.

Para los entrenamientos se utiliza una función adicional de Keras, siendo esta *EarlyStopping*[39], la cual permite detener el entrenamiento cuando la pérdida o una de las métricas ha dejado de mejorar. Esto permite realizar pruebas más controladas, a través de la configuración de sus múltiples parámetros.

Para cada una de las pruebas realizadas se introducen los valores de los hiperparámetros correspondientes y se comienza el entrenamiento. Cada entrenamiento de prueba tiene una duración de ejecución de entre 1 y 3 horas, en función de las épocas y del resto de hiperparámetros elegidos. Google Colab Pro ofrece diferentes tipos de GPUs y TPUs, pero debido a la naturaleza del modelo este debe entrenarse con la GPU A100, siendo esta la que más potencia y memoria ofrece de todas las disponibles. Con otro tipo de GPUs se aprecian errores de falta de memoria.

Antes de comenzar cada prueba, se crea un directorio nuevo en Google Drive para poder guardar los resultados del entrenamiento en diferentes archivos configurados con el módulo os[22]. Estos archivos conforman los pesos calculados por el modelo y el historial del modelo, donde quedan registrados todos los valores de la pérdida y las métricas en cada época, incluidos los valores de entrenamiento y validación. Junto a ellos se incluyen gráficas de dichos valores medidos, para poder analizar el comportamiento del entrenamiento. Al terminar el entrenamiento se utilizan la librería json[23] y la librería Matplotlib[21] para guardar todos los resultados al directorio creado.

Cabe destacar que, aunque las imágenes utilizadas para el entrenamiento y para la validación se introducen siempre en el mismo orden al modelo, la red no cuenta con una inicialización de los pesos por lo que dos pruebas realizadas utilizando los mismos parámetros pueden alcanzar resultados diferentes.

A continuación, se presentan los tres grandes grupos de experimentos realizados en función de la arquitectura del modelo. Esto se lleva a cabo llevando una metodología de trabajo iterativo, comenzando con una arquitectura simple y desarrollando ésta hasta alcanzar una estructura de la red compleja.

5.1. Primeros entrenamientos con un modelo simple

Para comenzar con las pruebas, se utiliza una versión algo más sencilla que la estructura del modelo mostrado previamente en la Figura 17. Esto se debe a que se pretende encontrar el modelo más sencillo con el cual obtener imágenes predichas correctas. De esta manera, la primera versión de la red utilizada no cuenta con la tercera capa del *bottleneck* ni con las diversas capas de *dropout*.

5.1.1. Resultados con un valor alto de lambda

Los primeros entrenamientos utilizan un valor de lambda muy alto, siendo este $\lambda = 100$. Se utilizan diferentes valores del *learning rate* que difieren entre 10^{-4} y 10^{-2} para observar las diferencias en el

efecto de este parámetro, llevando a cabo 4 entrenamientos. Estos reciben el nombre de Prueba 1, 2, 3 y 4 respectivamente.

Los resultados observados en todas las pruebas son muy similares, variando la velocidad de convergencia, tal y como es de esperar con los cambios de *learning rate*. En estos se ha encontrado que la pérdida desciende rápidamente, pero tras ello vuelve a crecer creando un pico y estancándose finalmente en un mismo valor. Con las métricas ocurre algo similar, comenzando con buenos valores, pero cayendo en picado en el momento en el que la pérdida crece. En la Figura 24 se observa un ejemplo de los resultados hallados en la prueba 4, utilizando un *learning rate* de valor 10^{-4}.

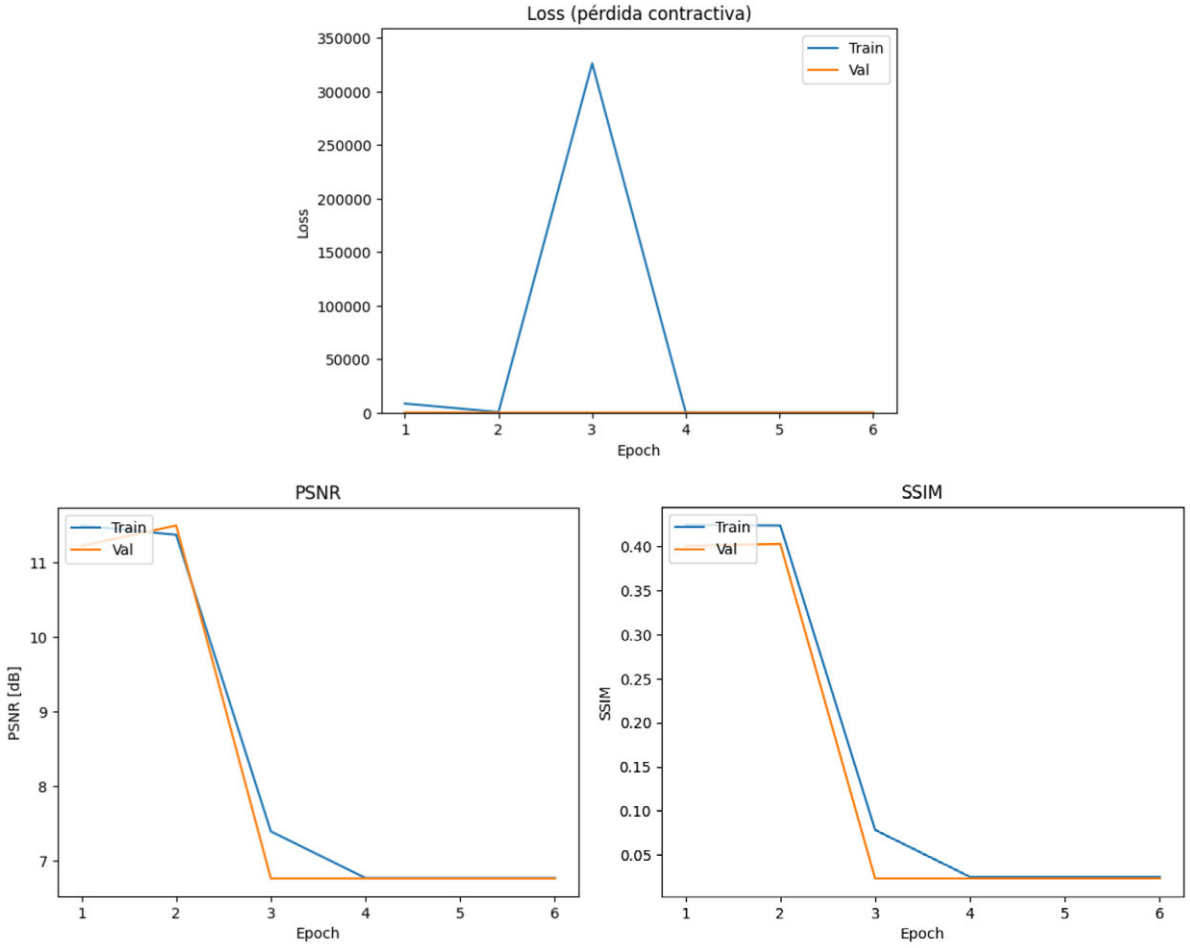

Figura 24: Resultados del entrenamiento de la prueba 4, con λ = 100 y un *learning rate* de 10^{-4}. Arriba: Valor de la pérdida. Abajo: Valor de las métricas PSNR y SSIM

Con esto se concluye que un valor muy alto de lambda conlleva una fuerte regularización, creando un modelo muy robusto y poco flexible que no es capaz de aprender características de las imágenes, por lo que las métricas caen en picado.

5.1.2. Resultados variando el valor de lambda

Para las siguientes pruebas se utiliza un valor moderado de lambda, realizando diferentes combinaciones de este junto con diferentes combinaciones del valor del *learning rate*. De esta manera, los valores utilizados de lambda varían entre 0,1 y 0,01, y para el *learning rate* se vuelven a utilizar valores entre 10^{-4} y 10^{-2}. Se realizan aquí un total de 9 pruebas, desde la prueba 5 hasta la 14.

Las pruebas se realizan utilizando *EarlyStopping*, controlando el valor de la pérdida y estableciendo la detención del entrenamiento si esta no consigue una mejora de al menos 0,1 en su valor con una paciencia de 3 épocas. Esto se implementa de manera que si la pérdida no desciende al menos un valor de 0,1 entre una época y la siguiente el modelo continúa el entrenamiento durante 3 épocas más, y si en ellas tampoco se reduce el valor el entrenamiento se detiene. Por esta razón, no todas las pruebas tienen el mismo número de épocas, ya que en algunas la pérdida converge de manera más rápida que en otras.

Se muestra un resumen de los resultados más interesantes en la Tabla 1. En ella se resaltan en color verde aquellas pruebas en las que se obtiene un valor prometedor de sus métricas.

Tabla 1: Resultados destacados de las métricas utilizando un modelo simple

Nº de prueba	Nº epochs	Lambda	Lrate	Pérdida	MSE	PSNR [dB]	SSIM	MS-SSIM
6	6	0,1	0,001	0,066	0,065	12,206	0,428	0,314
7	5	0,1	0,01	0,233	0,233	6,766	0,025	0,138
8	5	0,1	0,005	0,414	0,414	4,057	0,346	0,301
9	6	0,1	0,001	0,072	0,065	12,216	0,428	0,313
10	4	0,01	0,0001	0,233	0,233	6,766	0,025	0,138
11	6	0,01	0,001	0,065	0,065	12,221	0,428	0,313
12	5	0,01	0,01	0,233	0,233	6,766	0,025	0,138

En primer lugar, los valores de *learning rate* que coinciden con la obtención con mejores resultados han sido con un valor de 10^{-3}. Con ellos se obtiene un valor cercano de la pérdida, consiguiendo los valores más bajos entre todas las pruebas realizadas, destacando especialmente la prueba 6 y la 11. Además, las métricas obtenidas en estas pruebas son bastante prometedoras, ya que su valor se mantiene con el paso de las épocas e incluso muestra un ritmo de mejora. Para ilustrar esto se muestran los resultados del entrenamiento de la prueba 6 en la Figura 25, donde se observa como incluso los valores de validación son mejores que los de entrenamiento en métricas como el PSNR, y aunque en el SSIM y en el MS-SSIM estos se encuentran algo por debajo, estos están dentro de los límites esperados.

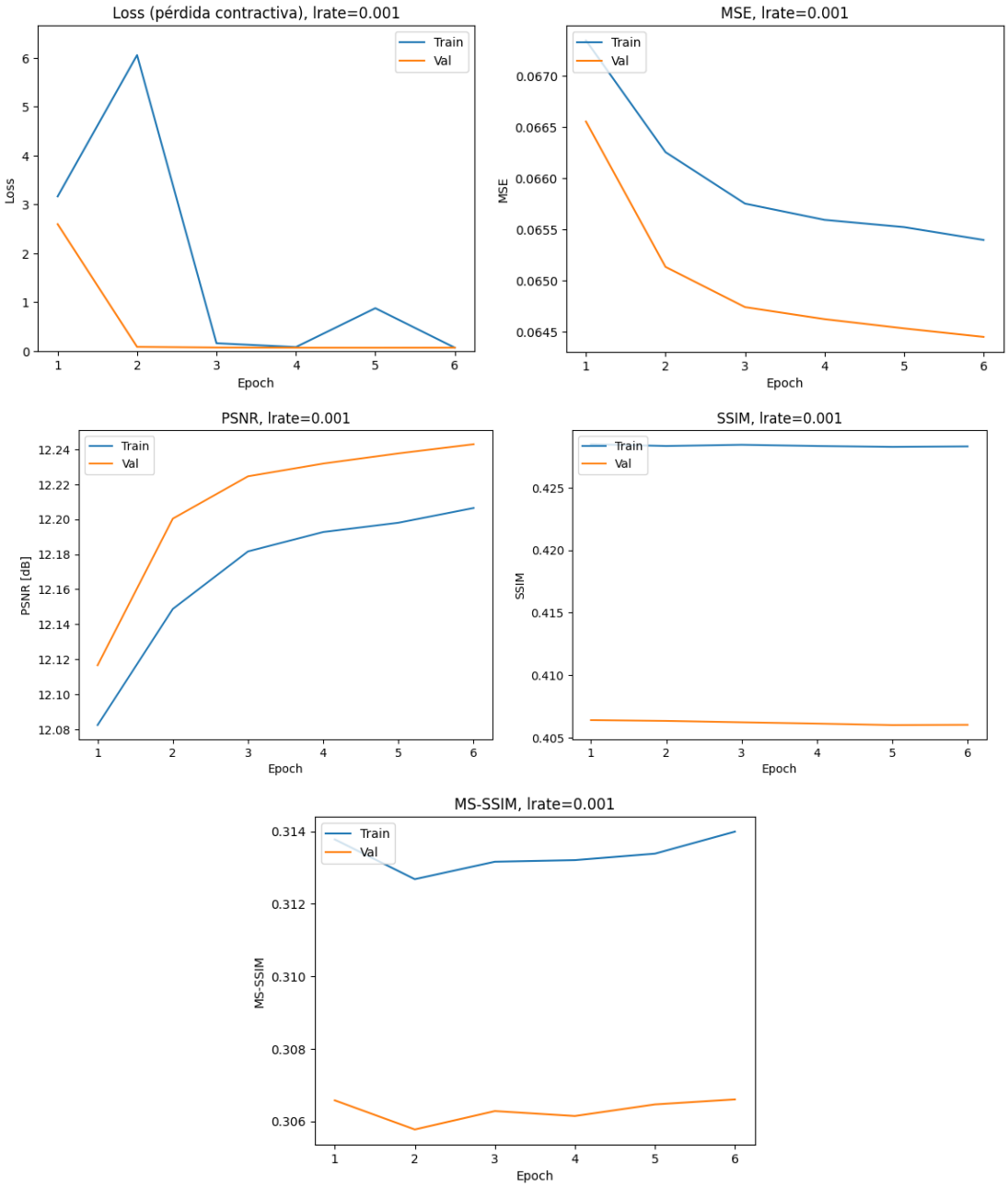

Figura 25: Resultados del entrenamiento de la prueba 6, con λ = 0,1 y un *learning rate* de 10^{-3}. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM

No se presentan los valores del entrenamiento de la prueba 11 al ser estos muy parecidos a las gráficas anteriores, donde también hay signos de mejora de las métricas con el tiempo.

Tras la obtención de los buenos resultados de las métricas se realizan predicciones con los modelos entrenados de las pruebas 6 y 11, realizando en primer lugar la evaluación de estos. Los resultados de la evaluación se presentan en la Tabla 2.

Tabla 2: Resultados de la evaluación de los valores de los modelos de las pruebas 6 y 11

Prueba	Pérdida	MSE	PSNR [dB]	SSIM	MS-SSIM
6	0,065	0,063	12,354	0,439	0,321
11	0,063	0,063	12,364	0,438	0,321

Tras representar las imágenes predichas, en ambas se perciben imágenes completamente grises a simple vista. Se estudia el valor mínimo y máximo de dichas imágenes y se aprecia que en las de la prueba 6 el valor mínimo de los píxeles se encuentra en 105 y el máximo en 117, mientras que en las imágenes predichas por el modelo de la prueba 11 el valor mínimo se encuentra en 100 y el máximo en 112. Por este motivo las imágenes se ven de color gris aparentemente uniforme a simple vista. Sin embargo, si se ajusta la función de representación de la imagen, siendo esta *imshow* [40] de Matplotlib, de manera que se represente la imagen con respecto a sus mínimos y máximos propios se pueden distinguir ciertas estructuras en la imagen, especialmente en la imagen predicha por la prueba 6. En la Figura 26 se presentan todas las imágenes predichas obtenidas, donde las de la columna izquierda corresponden a la prueba 6 y las de la columna derecha a la prueba 11.

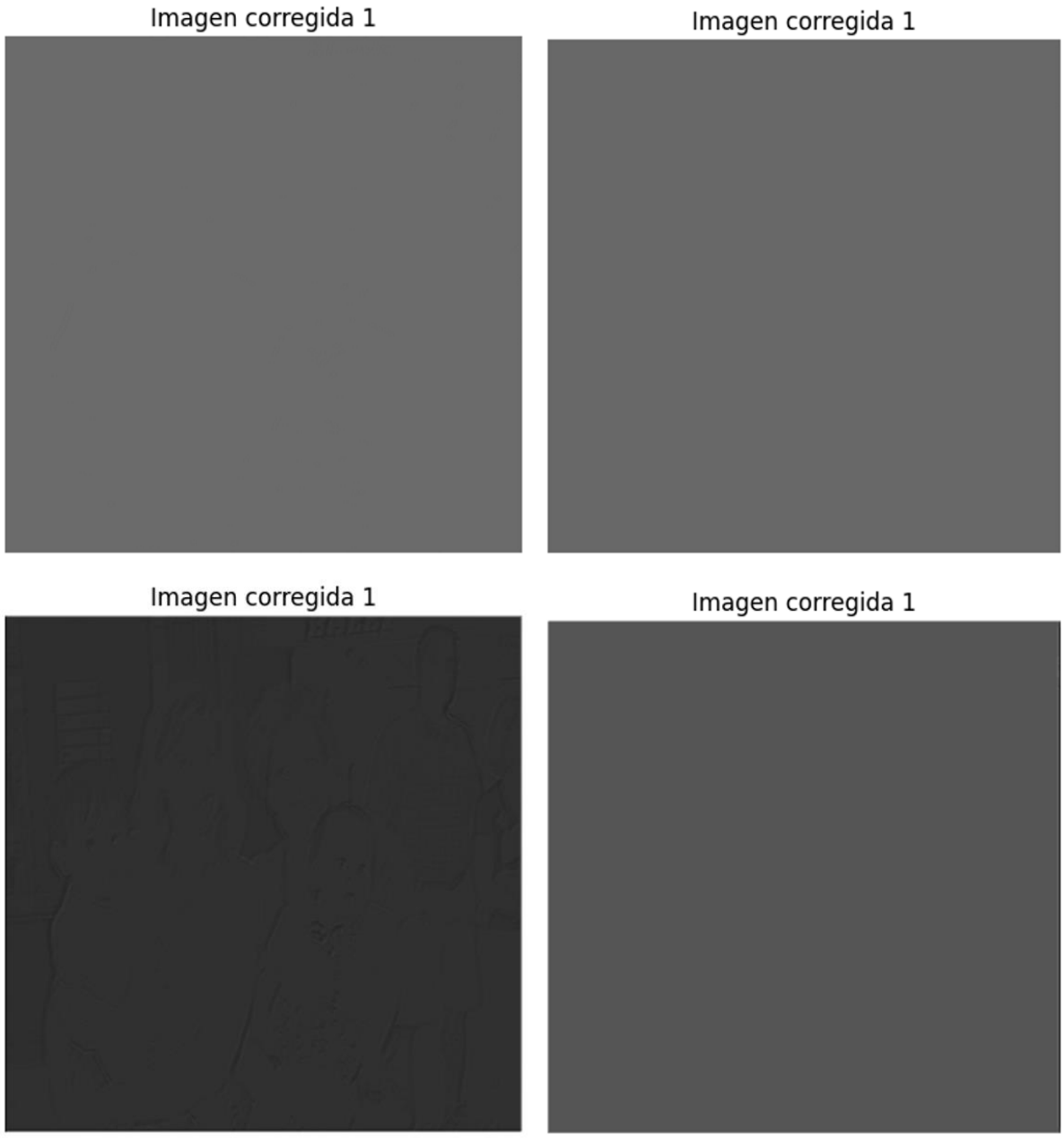

Figura 26: Imágenes predichas por los modelos de las pruebas 6 y 11. Arriba: Representación real de las imágenes. Abajo: Representación en función de los valores mínimos y máximos de los píxeles

Para analizar mejor los resultados obtenidos se ofrece la imagen *ground truth* correspondiente para las entradas en la Figura 27.

Figura 27: Imagen *ground truth* de las imágenes predichas

Comparando los resultados, la imagen predicha por la prueba 6 ofrece ligeramente las siluetas de las personas que aparecen en la fotografía, por lo que se comprueba el correcto funcionamiento de las *skip connections* ya que se mantienen así las localizaciones espaciales. Esto no se percibe en la imagen predicha por la prueba 11, pero podría ser porque el número de épocas utilizado en ambas pruebas ha sido muy bajo, habiendo realizado solo 6 épocas ya que se buscaba un primer acercamiento a la solución.

En los casos en los que se ha utilizado un valor del *learning rate* de 10^{-2}, siendo un ejemplo la prueba 7, la pérdida y las métricas se han estancado rápidamente. En la Figura 28 se ofrecen los resultados completos de esta prueba, donde se observa claramente el momento donde la pérdida se estanca en un mismo valor y con ello las métricas caen en picado, estancándose junto a la pérdida.

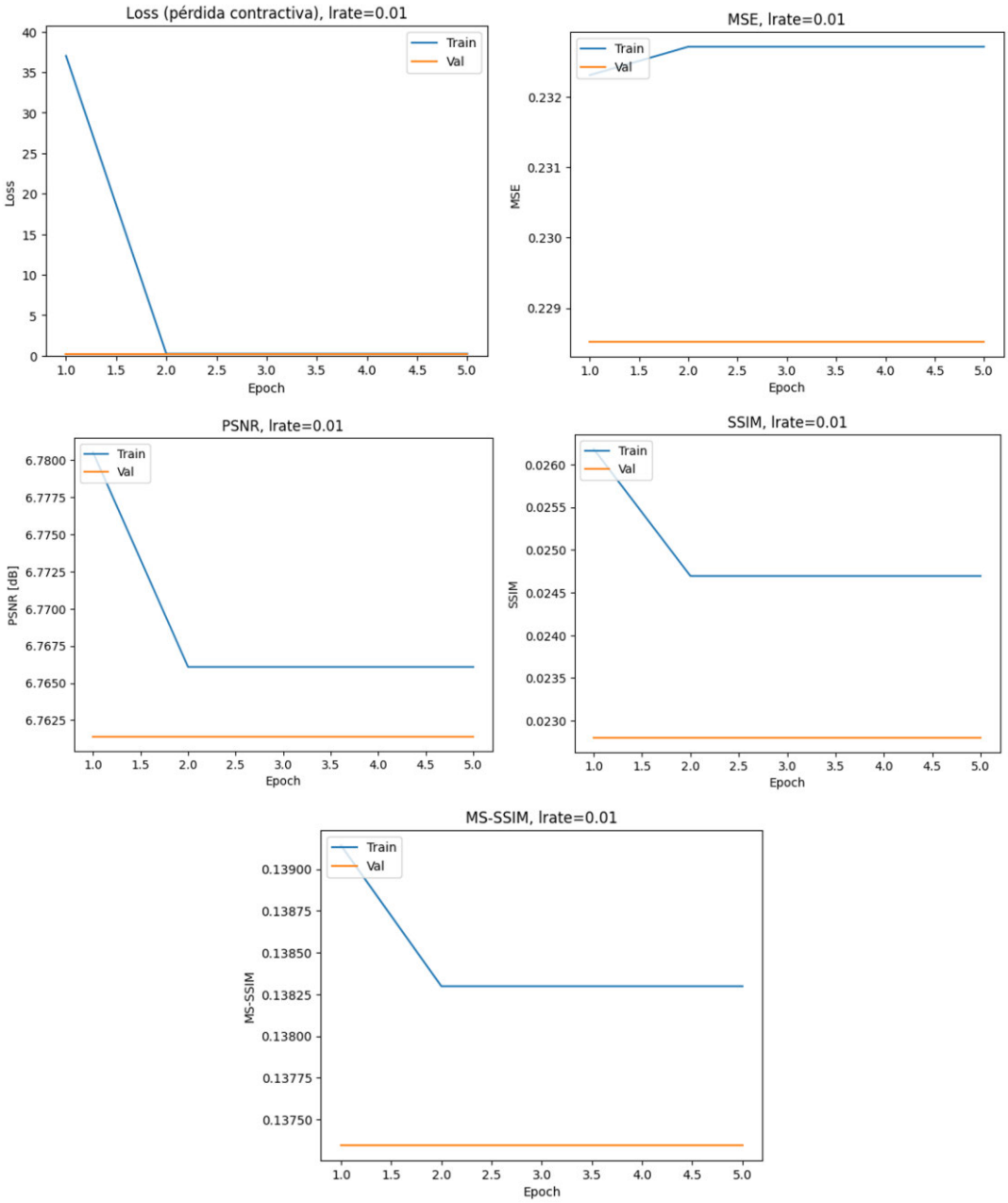

Figura 28: Resultados del entrenamiento de la prueba 7, con λ = 0,1 y un *learning rate* de 10^{-2}. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM

Estos resultados son importantes de analizar, ya que en otras pruebas realizadas con hiperparámetros diferentes se han repetido el estancamiento de la pérdida y la bajada y estancamiento de las métricas. Siempre que ocurre este fenómeno se puede comprobar que todos los valores finales de la pérdida y las métricas coinciden, tal y como se aprecia en los resultados ofrecidos anteriormente en la Tabla 1 para las pruebas 7, 10 y 12. Esto indica que podría haber un mínimo local que ofrece estos valores del que la función de pérdida no consigue retroceder. También podría tratarse de un caso de explosión de gradientes, debido a la súbita subida que sufre el modelo. Esta subida ocurre tanto desde un principio, como se ha visto en los resultados de la prueba 7, como tras varias épocas de entrenamiento, lo cual ocurre en la prueba 12. Se ofrece una comparación de varios casos de este fenómeno en la Figura 29.

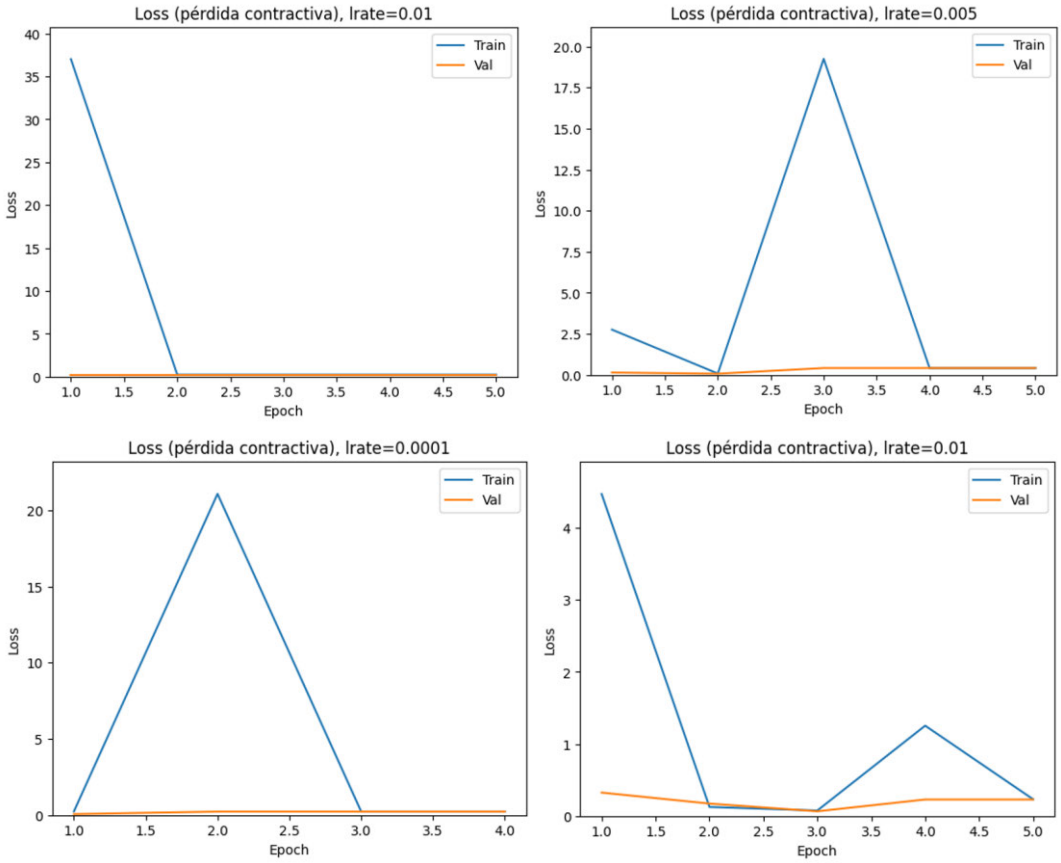

Figura 29: Comparativa de los resultados de subida repentina de la pérdida. Arriba izquierda: Prueba 7. Arriba derecha: Prueba 8. Abajo izquierda: Prueba 10. Abajo derecha: Prueba 12

Para comprobar si este fenómeno está relacionado directamente con el valor de los hiperparámetros, se realizan las pruebas 13 y 14. En ellas se replican las condiciones de otras pruebas satisfactorias anteriores, repitiendo así las pruebas 9 y 6 respectivamente, pero esta vez retirando el *EarlyStopping* de manera que el modelo pueda desarrollarse durante 20 épocas para ver su evolución. El resultado en ambas pruebas ha sido la aparición del fenómeno mencionado, observando la subida de la pérdida alrededor de la época número 10. De esta manera se descarta la dependencia directa del valor de los hiperparámetros, y se busca una variación de la arquitectura de la red para ver si con ello se puede solucionar la subida súbita de la pérdida.

5.2. Pruebas modificando la arquitectura del modelo

Se modifica la arquitectura del modelo para ver si con ello se consiguen evitar los resultados. Para ello se añade una tercera capa en el bottleneck y se realiza un entrenamiento de prueba con los mismos valores de los hiperparámetros que en la prueba 11. Como esta vez se busca entrenar el modelo sin que ocurra el decaimiento de las métricas, se utiliza *EarlyStopping* vigilando el valor del PSNR con una paciencia de 4 épocas y estableciendo que se guarden los pesos de la época con mejores resultados. Esta prueba recibe el nombre de prueba 15.

Los resultados que se consiguen en la prueba 15 son también prometedores y se muestran en la Figura 30. En ellos la pérdida sufre una subida en la época 6, pero tras ello vuelve a los valores óptimos que había conseguido antes de dicha subida. Como se está monitoreando el PSNR y este se mantiene estable a partir de la época 7, la red detiene el entrenamiento en la época 11 y guarda los pesos de la época 7 ya que en esta se consigue el valor más alto del PSNR.

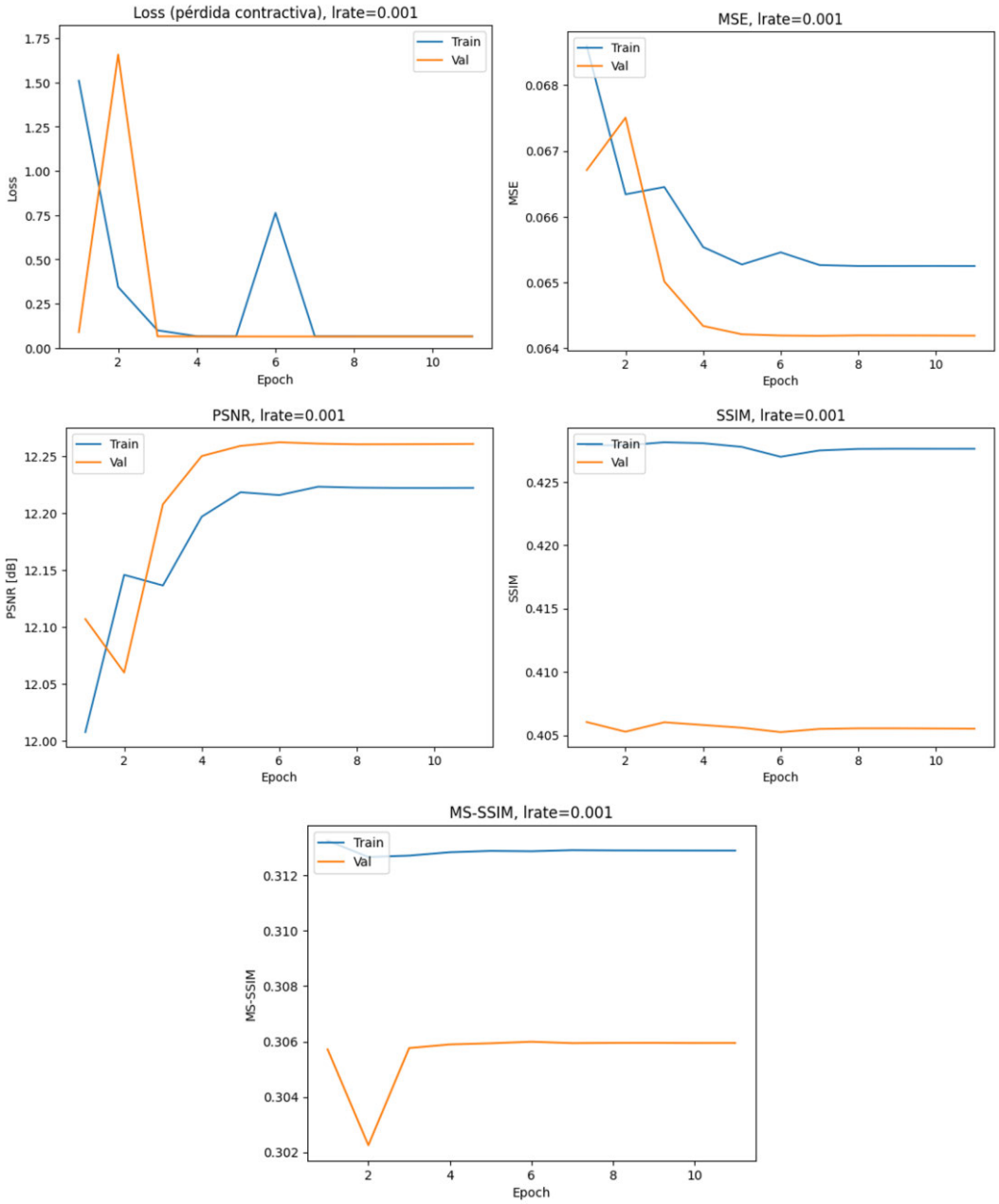

Figura 30: Resultados del entrenamiento de la prueba 15, con λ = 0,01 y un *learning rate* de 10^{-3}. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM

Tras el entrenamiento se calculan las imágenes predichas, obteniendo al evaluar el modelo a través de *evaluate* valores muy prometedores. Estos se muestran en la Tabla 3.

Tabla 3: Resultados de la evaluación de los valores del modelo de la prueba 15

Pérdida	**MSE**	**PSNR [dB]**	**SSIM**	**MS-SSIM**
0,063	0,063	12,364	0,438	0,321

Sin embargo, al representar las imágenes predichas estas tienen un color grisáceo aparentemente uniforme, tal y como se aprecia en la Figura 31. En ella se muestran tanto la representación real como la representación en función de sus valores máximos y mínimos. Las imágenes predichas esta vez

tienen un rango mayor de valores, siendo el valor mínimo de los píxeles de 90 y el máximo de 127, pero en esta ocasión no se distinguen formas significativas como en las pruebas anteriores.

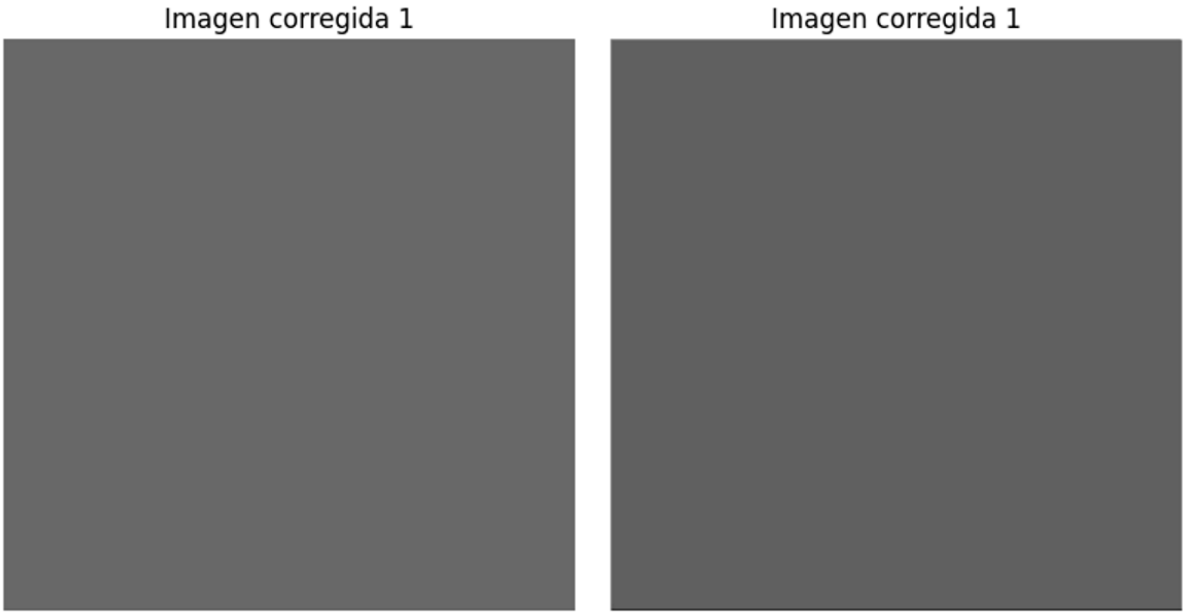

Figura 31: Imágenes predichas por el modelo de la prueba 15. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles

De esta manera, aunque se consiguen valores bastante alentadores de las métricas y de la pérdida, las imágenes predichas no se acercan al resultado esperado. Esto podría deberse a que como la red se queda con los pesos de la época número 7 al ser los que mejor valor del PSNR presentan, el modelo podría estar poco entrenado ya que se trata de un número algo bajo de épocas. Debido a esto, se realiza otra prueba con los mismos parámetros que la anterior, pero sin utilizar *EarlyStopping* para que el modelo pueda ser entrenado durante 20 épocas sin pausa, numerando esta prueba como prueba 15 bis.

La prueba 15 bis se desarrolla durante 20 épocas, y comienza de manera muy parecida a la prueba 15, con valores similares a esta. Sin embargo, en la época 8 sufre una subida muy alta de la pérdida tras la cual este parámetro no recupera los valores óptimos que había alcanzado previamente, ya que tras la subida vuelve a bajar y se estanca en los valores ya vistos en las pruebas 7, 10 o 12. Las métricas también sufren el mismo efecto estudiado en los casos anteriores. Además, tanto el valor de la pérdida como el de las métricas se mantiene igual, sin variación alguna, durante el resto del entrenamiento. Por lo tanto, se verifica que, si la pérdida se estanca en dicho mínimo relativo, el modelo no consigue salir de este, ni siquiera con el cambio de la arquitectura. Si se obtienen imágenes predichas con el modelo 15 bis, al representarlas se observan completamente negras ya que todos sus píxeles tienen el valor 0.

De esta manera se verifica que la arquitectura actual no es suficiente y que se necesitan medidas para evitar la subida súbita de la perdida y la caída de las métricas.

5.3. Pruebas con técnicas de *clipping*

En el aprendizaje automático se define el *clipping* como una técnica utilizada para establecer un rango específico dentro del cual se limitan los valores de un gradiente, un conjunto de datos o una función.

Esto es utilizado especialmente para evitar problemas con dichos valores, eliminando así la posibilidad de que estos puedan volverse demasiado grandes o pequeños, pudiendo con ello causar ciertos problemas o efectos no deseados en el entrenamiento del modelo.

Dado las subidas súbitas en la pérdida que se han estado observando, es posible que esté ocurriendo el fenómeno de explosión de gradientes en el modelo, el cual es corriente en redes profundas. Esto ocurre si los gradientes se vuelven extremadamente grandes durante el proceso de retropropagación, y puede causar que los pesos de la red alcancen valores muy altos, lo cual supone inestabilidad en el modelo de manera que este no puede aprender de manera adecuada.

Para tratar los posibles problemas de explosión de gradiente en el modelo, se utilizan varias técnicas de *clipping* con diferentes alcances y se analizan los resultados. Además, se añaden a la arquitectura del modelo anterior capas de *dropout*, quedando así el modelo final que se ha presentado previamente en la Figura 17. Esto se lleva a cabo para aportar cierta regularización que podría ayudar al modelo a evitar las subidas súbitas, además de ayudar también a evitar los estancamientos y el *overfitting* en caso de que el problema se debiera a ello, lo cual tampoco es descartado. El *dropout* establecido en las capas tiene un valor de 0,5.

En total se aplican 3 técnicas de *clipping* diferentes, expuestas a continuación junto a sus resultados.

5.3.1. Primera técnica: *weight clipping*

En primer lugar, se utiliza la técnica de *weight clipping*, la cual consiste en la limitación de los pesos de las capas de manera directa para que estos se encuentren dentro de un rango concreto. Esta es la técnica que más restringe al modelo entre todas las utilizadas, ya que se verifica la red peso por peso tras cada época y en caso de que se superen los márgenes establecidos el peso recibe el valor límite.

Para poder llevar esto a cabo se crea una clase auxiliar que hereda de la clase *Callback*[41] (a la cual pertenecen funciones como *EarlyStopping*), de manera que la comprobación de los pesos pueda realizarse al final de cada época a lo largo de todo el entrenamiento. Esta recibe el nombre de *WeightClippingCallback*, y se puede observar su código en el anexo A.3. En ella se comienza creando una función de inicialización, en la cual se establecen los valores mínimos y máximos que se quieren utilizar para los pesos. Después, como otra función independiente, se utiliza el método *on_train_batch_end* de *Callback* para poder definir la operación a realizar al finalizar cada época. En esta función se recorren las capas del modelo, de las cuales se obtienen los pesos, y estos se limitan según el rango escogido. Tras ello se introducen los pesos ya limitados de nuevo en las capas.

Para introducir esta clase en el entrenamiento del modelo se inicializa como variable añadiendo los límites deseados y se introduce como *Callback* al utilizar el método *fit*. Con estos nuevos cambios se realizan 5 pruebas, desde la prueba 16 hasta la prueba 20.

En la prueba 16 se ha utilizado un *learning rate* de 10^{-3}, junto con un valor de lambda de 0,01, habiendo conseguido con ambos valores buenos resultados de las métricas en las pruebas 11 y 15. Para el *clipping* se ha escogido un rango de [-0’1, 0’1], y se han utilizado 13 épocas sin *EarlyStopping* para comprobar el funcionamiento en un periodo fijo sin pausas. El resultado se muestra en la Figura 32, donde se puede observar como la pérdida se mantiene a lo largo del tiempo, aunque en la época 11 hay un ligero aumento que luego vuelve a disminuir. Durante este entrenamiento se consiguen valores del PSNR bastante altos, hasta la época 11 donde estos decaen, sin embargo, el SSIM y el MS-SSIM constan con valores algo más bajos de lo normal en la gran mayoría de las épocas y en la época 11

sufren un gran aumento. Aunque estos valores sean algo inusuales en comparación con las pruebas anteriores, el hecho de que las métricas no hayan sufrido un descenso en picado hasta la mitad de su valor original como se ha visto anteriormente es un buen indicador. Además, el SSIM y el MS-SSIM son indicadores que se acercan más a la capacidad de análisis del SVH, de manera que es más interesante obtener resultados mejores en estas métricas.

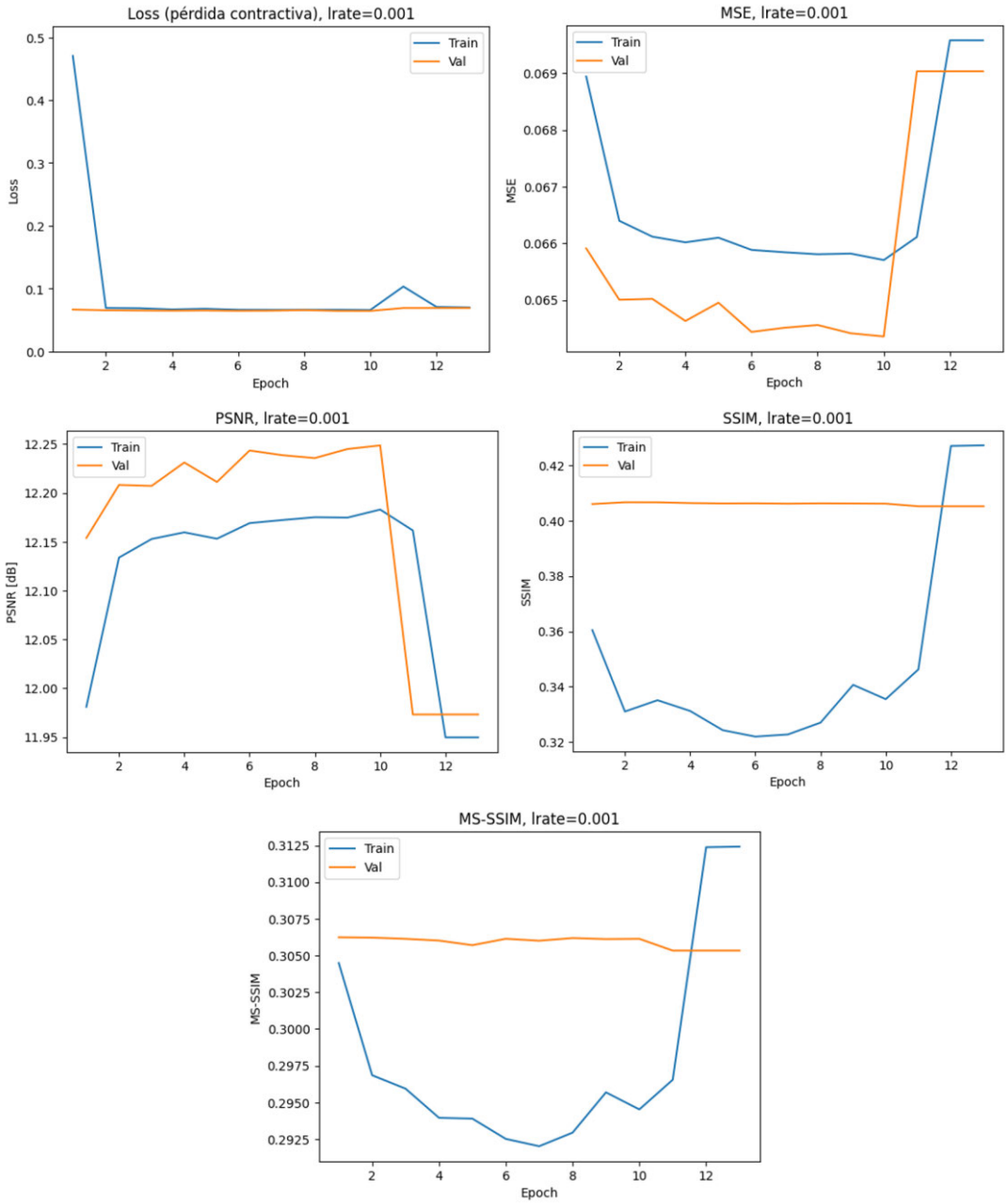

Figura 32: Resultados del entrenamiento de la prueba 16, con λ = 0,01, un learning rate de 10^{-3} y *weight clipping* de ±0,1. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM

Tras visualizar los resultados del entrenamiento se evalúa el modelo, obteniendo tras ello buenos valores de las métricas que se presentan en la Tabla 4.

Tabla 4: Resultados de la evaluación de los valores del modelo de la prueba 16

Pérdida	MSE	PSNR [dB]	SSIM	MS-SSIM
0,067	0,067	12,108	0,438	0,321

Tras ello se calculan las imágenes predichas y se representan. En este caso, el valor mínimo y el valor máximo de los pixeles obtienen el mismo valor, siendo este de 121, por lo que las imágenes predichas se observan completamente grises. Esto sorprende debido a los buenos resultados de las métricas, aunque cabe destacar que el valor de la pérdida y el valor del MSE es el mismo por lo que la parte contractiva de la pérdida podría no estar afectando en las imágenes predichas. Esto coincide con algunos resultados de pruebas anteriores, por lo que podría tratarse realmente de un caso de *overfitting*, aunque no se descartan otros fenómenos.

Para tratar de mitigar este efecto y para comprobar cómo puede afectar una mayor restricción en los pesos se realiza una nueva prueba, siendo esta la prueba 17, en la que se ajustan algo más los pesos de la red, utilizando así un rango de [-0’05, 0’05]. Para ello se utilizan los mismos valores para el *learning rate* y para lambda, utilizando esta vez 20 épocas a través de las que se puede estudiar mejor la evolución de las métricas. Los resultados obtenidos se presentan en la Figura 33. En ella se observan como esta vez los resultados son similares a los de la prueba anterior, aunque esta vez no se aprecian subidas o bajadas súbitas de las métricas. La pérdida se mantiene constante y los resultados de todos los valores, tanto de entrenamiento como de validación, son bastante positivos. Tras ello se evalúa el modelo, obteniendo los valores mostrados en la Tabla 5, los cuales son también bastante similares a los de la prueba 16.

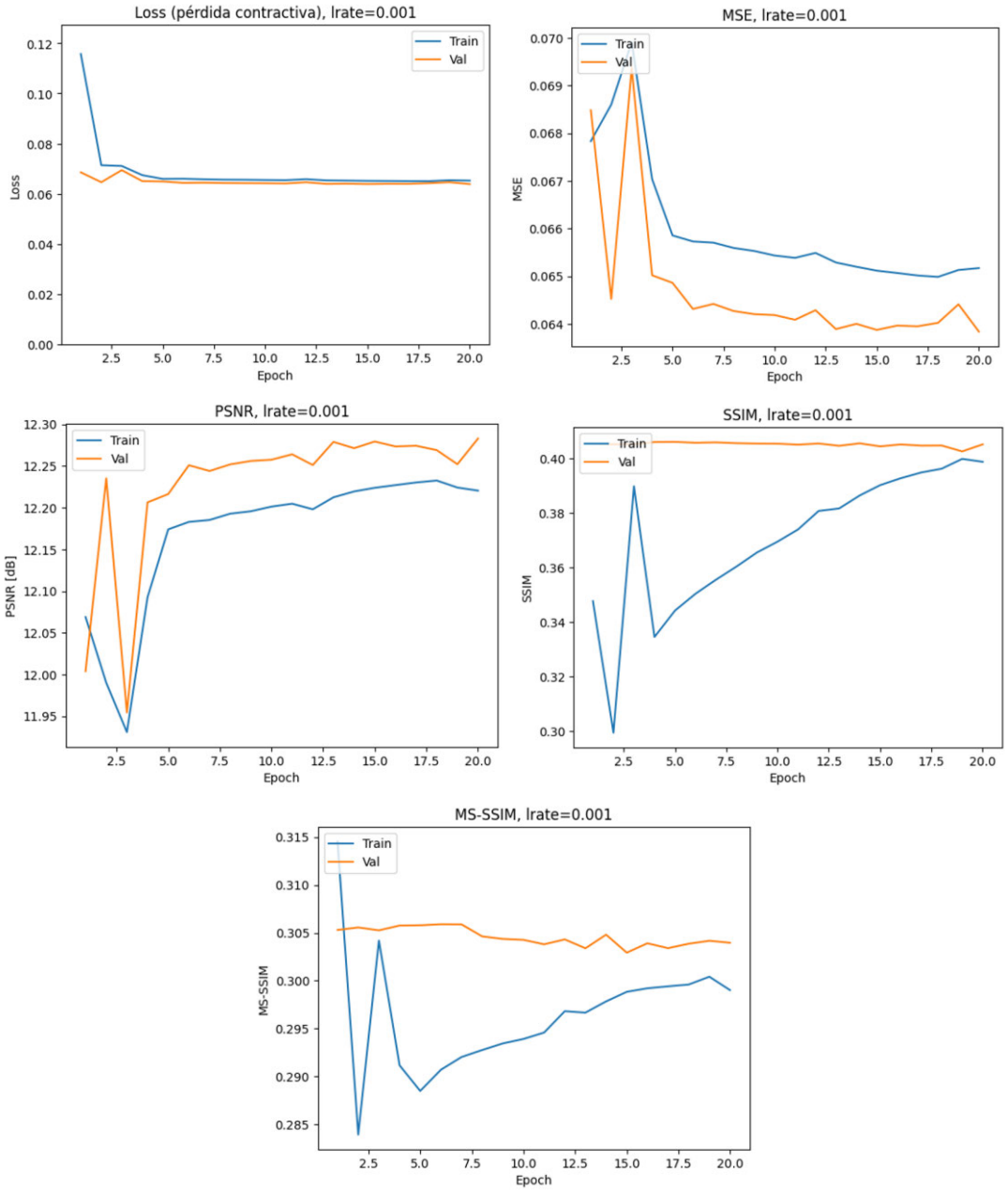

Figura 33: Resultados del entrenamiento de la prueba 17, con λ = 0,01, un learning rate de 10^{-3} y *weight clipping* de ±0,05. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM

Tabla 5: Resultados de la evaluación de los valores del modelo de la prueba 17

Pérdida	**MSE**	**PSNR [dB]**	**SSIM**	**MS-SSIM**
0,063	0,063	12,390	0,438	0,319

En la Figura 34 se muestran las imágenes predichas por el modelo de la prueba 17. En ellas, el valor mínimo de los píxeles es de 92 y el máximo de 118. Sin embargo, todas las imágenes obtenidas tienen la misma apariencia, con dos franjas de tonos diferentes en la parte superior e inferior de la imagen. Si se representa la imagen en función de sus valores máximos y mínimos, dichas franjas se ven muy claras y muy oscuras, por lo que ahí es donde se concentran los píxeles de mayor y menor valor. Sin embargo, no se distingue ningún tipo de estructura referente a la fotografía original.

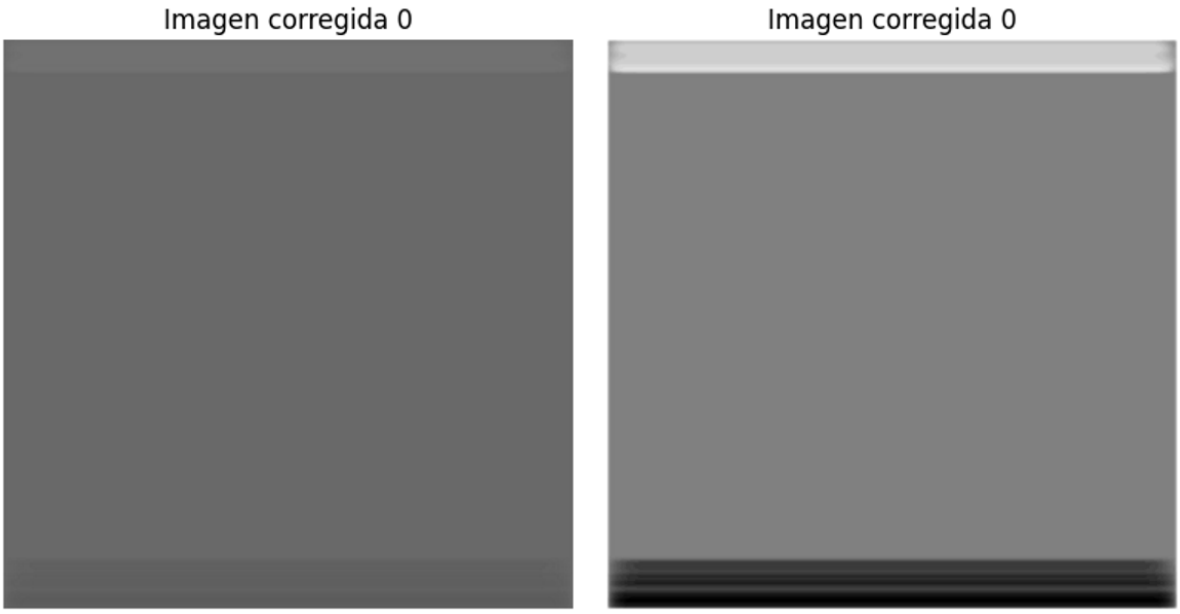

Figura 34: Imágenes predichas por el modelo de la prueba 17. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles

La conclusión que se toma tras estas pruebas reside en que la técnica de *weight clipping* es demasiado restrictiva para este modelo, ya que la red no consigue aprender ninguna de las características buscadas. Como al modelo se han añadido capas de *dropout* además de restringir los pesos, se realizan tres pruebas adicionales eliminando el *weight clipping* para comprobar el funcionamiento de la red con la influencia del *dropout*. Estas consisten en las pruebas 18, 19 y 20, y se usan los mismos parámetros de las pruebas anteriores, utilizando 15 épocas y *EarlyStopping* configurado para guardar los pesos de los mejores resultados, vigilando la pérdida.

En la primera prueba el modelo se estanca con los valores ya conocidos. Se realiza otro entrenamiento, la prueba 19, por si los pesos iniciales anteriores no hubieran sido los adecuados, y esta vez el modelo se mantiene estable hasta la época 12, tras la cual sufre el fenómeno de subida y estancamiento ya estudiado. El modelo guarda los pesos de la época 12. Se ofrecen los resultados del entrenamiento de la prueba 19 en la Figura 35, donde se observa que los valores de la pérdida, del MSE y del PSNR hasta la época 12 son muy similares a otros vistos en pruebas anteriores. Sin embargo, los valores del SSIM y del MS-SSIM tienen formas diferentes en la etapa del entrenamiento, mientras que los de validación son bastante similares a los vistos anteriormente. Esto significa que el *dropout* afecta más a estas dos métricas.

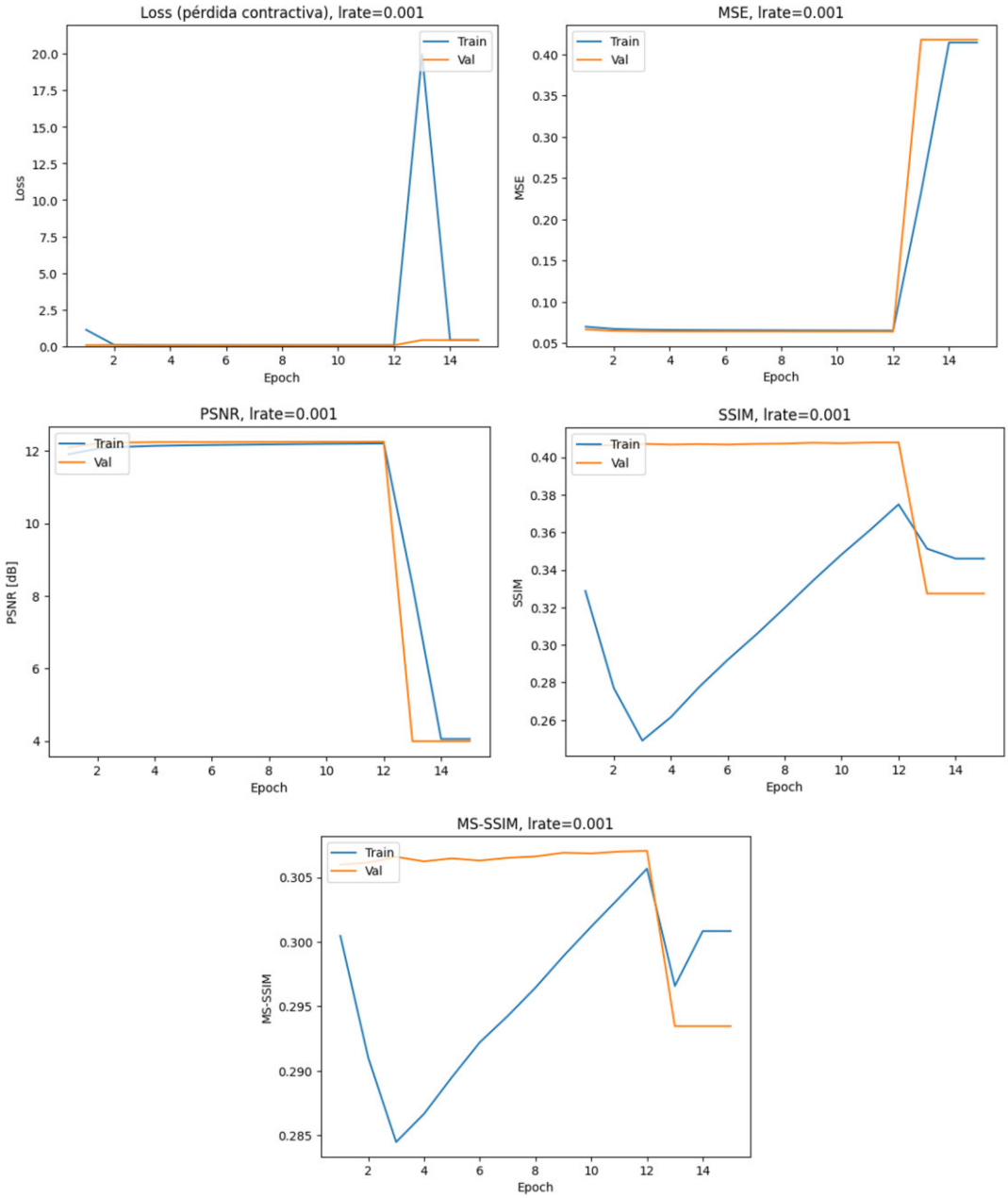

Figura 35: Resultados del entrenamiento de la prueba 19, con λ = 0,01 y un learning rate de 10^{-3}. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM

Tras esto, se evalúa el modelo obteniendo los valores de las métricas que se observan en la Tabla 6. Estas exponen un valor bastante elevado de SSIM y de MS-SSIM.

Tabla 6: Resultados de la evaluación de los valores del modelo de la prueba 19

Pérdida	MSE	PSNR [dB]	SSIM	MS-SSIM
0,063	0,063	12,363	0,440	0,322

En la Figura 36 se aprecian las imágenes predichas calculadas. Esta vez el valor de los píxeles oscila entre 104 y 114, pero incluso con esta ligera variación se pueden apreciar claramente los bordes de las personas y objetos que figuran en la fotografía si esta se representa en función de sus valores mínimos y máximos. Mientras que en otras pruebas anteriores las imágenes predichas son todas iguales a

simple vista (incluso utilizando ambos tipos de representación), en esta prueba se obtienen imágenes predichas claramente diferenciadas al observarlas con la representación en función de los valores límite. Por ello, se ofrecen más ejemplos de resultados en la Figura 37. Todas estas imágenes tienen el mismo valor mínimo de 104 y valor máximo de 114.

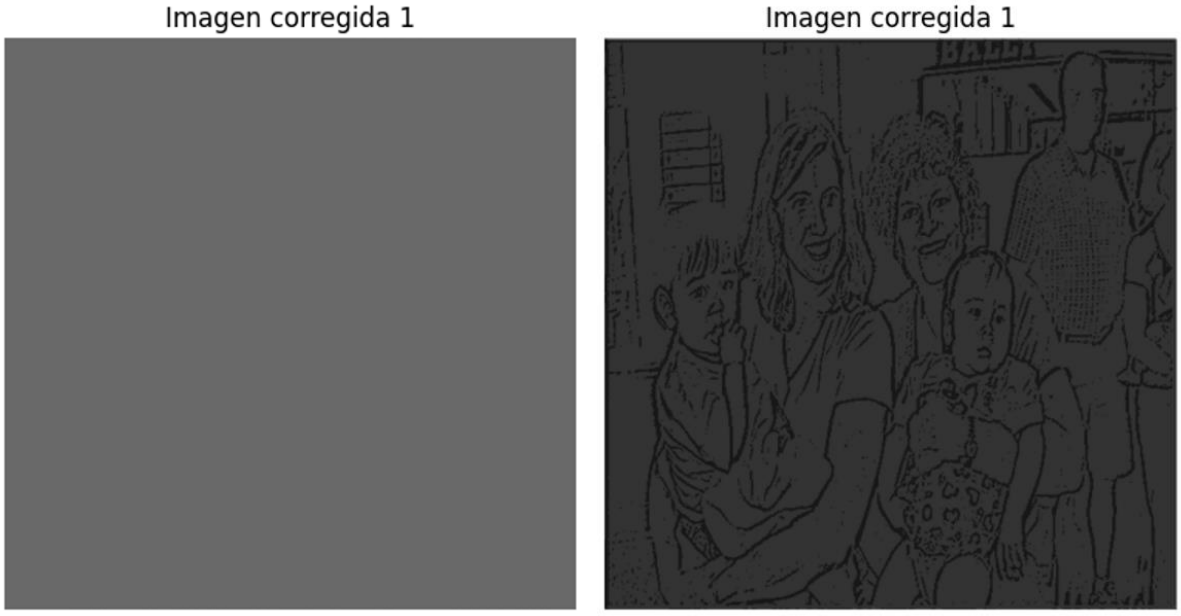

Figura 36: Imágenes predichas por el modelo de la prueba 19. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles

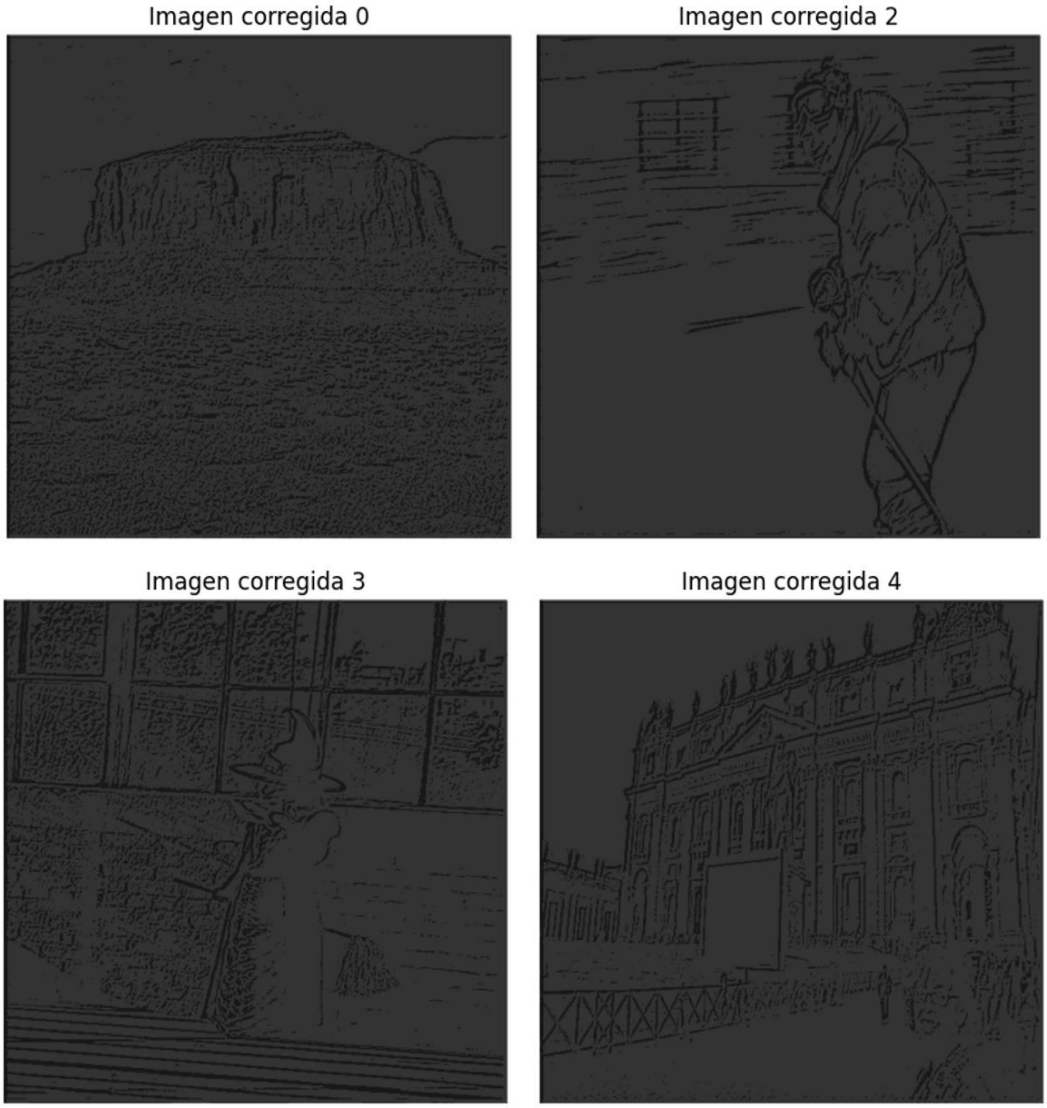

Figura 37: Otras imágenes predichas por el modelo de la prueba 19, representadas en función de sus valores mínimos y máximos

La prueba 20 trata de obtener de nuevo los valores prometedores de la anterior, pero tal y como ocurre con la 18 se estanca tras 6 épocas. Esto quiere decir que aunque el *dropout* mejore los resultados, este no es suficiente para evitar las subidas de la pérdida y el estancamiento de las métricas, por lo que el modelo necesita de la aplicación de otras técnicas para evitar dichos fenómenos. Como el *weight clipping* ha demostrado ser demasiado restrictivo, se estudian otros tipos de *clipping* diferentes.

5.3.2. Segunda técnica: *clipnorm*

Para las pruebas 21 y 22 se prueba utilizar el atributo *clipnorm* del optimizador *Adam* [34]. Este atributo permite limitar la norma de todos los gradientes, siendo esta la raíz cuadrada de la suma de los cuadrados de los gradientes. Si la norma total excede el valor especificado, los gradientes se escalan proporcionalmente para que la norma total sea igual a ese valor. También se añade el atributo *weight_decay* en *Adam* [34] para ayudar con la regularización.

El tipo de *clipping* que trata *clipnorm* tiende a ser más suave que el *weight clipping*, aunque depende del valor utilizado. Para incluirlo simplemente se añade este como un nuevo parámetro al inicializar el optimizador *Adam*.

Con esta técnica se realizan dos pruebas, la prueba 21 y la 22. Cada una de estas pruebas utiliza 20 épocas con *EarlyStopping* vigilando la pérdida y guardando los pesos de las mejores épocas. Para ambas se utiliza un valor de lambda de 0,01 y un *learning rate* de 10^{-4}. Este valor se ha reducido en comparación a pruebas anteriores para poder apreciar mejor las actualizaciones de la pérdida y de las métricas en cada época. Para la primera prueba se utiliza un valor de *clipnorm* de 0,5 y para la segunda un valor de *clipnorm* de 1. Además, en ambas pruebas se utiliza un valor de *weight_decay* de 10^{-5}.

Se muestran en primer lugar los resultados del entrenamiento de la prueba 21 en la Figura 38, y rápidamente se puede comprobar que estos son muy diferentes a otros entrenamientos anteriores. Lo que más destaca es que las métricas de entrenamiento son de las mejores que se han obtenido, especialmente en el caso de la pérdida, del MS-SSIM o del PSNR, sin embargo, el valor de estas en la validación es muy diferente y aumenta con el paso de las épocas. También sorprende que con los valores tan altos de las métricas que se observan, el valor del SSIM no destaca e incluso consigue un mejor valor de validación.

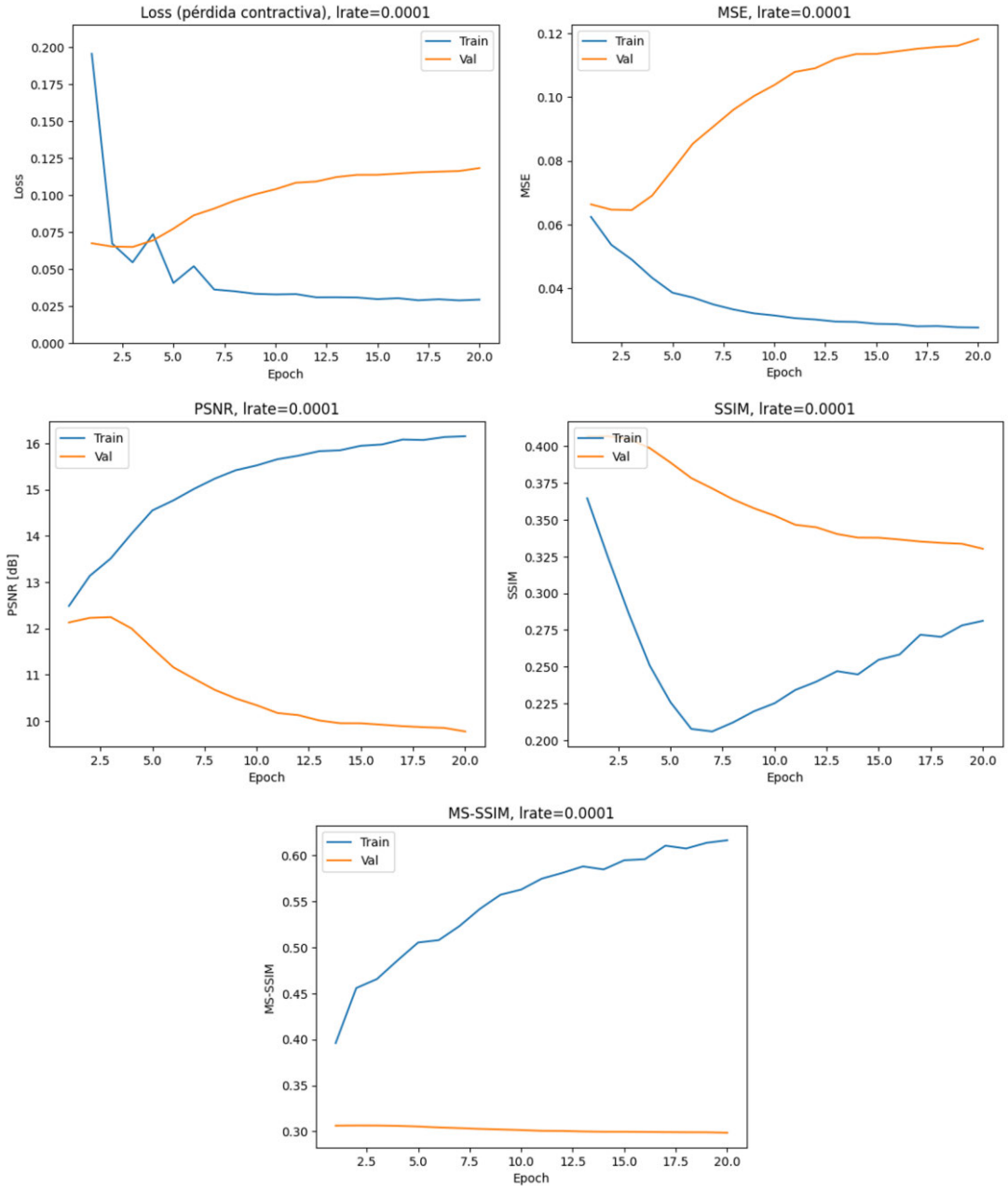

Figura 38: Resultados del entrenamiento de la prueba 21, con λ = 0,01, un learning rate de 10^{-4} y *clipnorm* de 0,5. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM

Este tipo de comportamiento parece un claro ejemplo de *overfitting*, debido a la gran diferencia de valores entre las métricas de entrenamiento y de validación, y esto se refleja a la hora de evaluar el modelo consiguiendo los valores mostrados en la Tabla 7. Aun habiendo conseguido resultados excelentes en el entrenamiento, alcanzando valores de la pérdida de 0,029 o valores del MS-SSIM superiores a 0,60, esta técnica no parece ser la correcta para entrenar este tipo de modelo, ya que todas las métricas consiguen valores peores que en pruebas anteriores.

Tabla 7: Resultados de la evaluación de los valores del modelo de la prueba 21

Pérdida	**MSE**	**PSNR [dB]**	**SSIM**	**MS-SSIM**
0,118	0,118	9,862	0,356	0,312

Esto también afecta a las imágenes predichas. En la Figura 39 se muestra la imagen, cuyo valor mínimo se encuentra ahora en 45 y el máximo en 77. En esta ocasión no se distingue ningún tipo de estructura o forma, ni siquiera al representar la imagen respecto a su rango de valores.

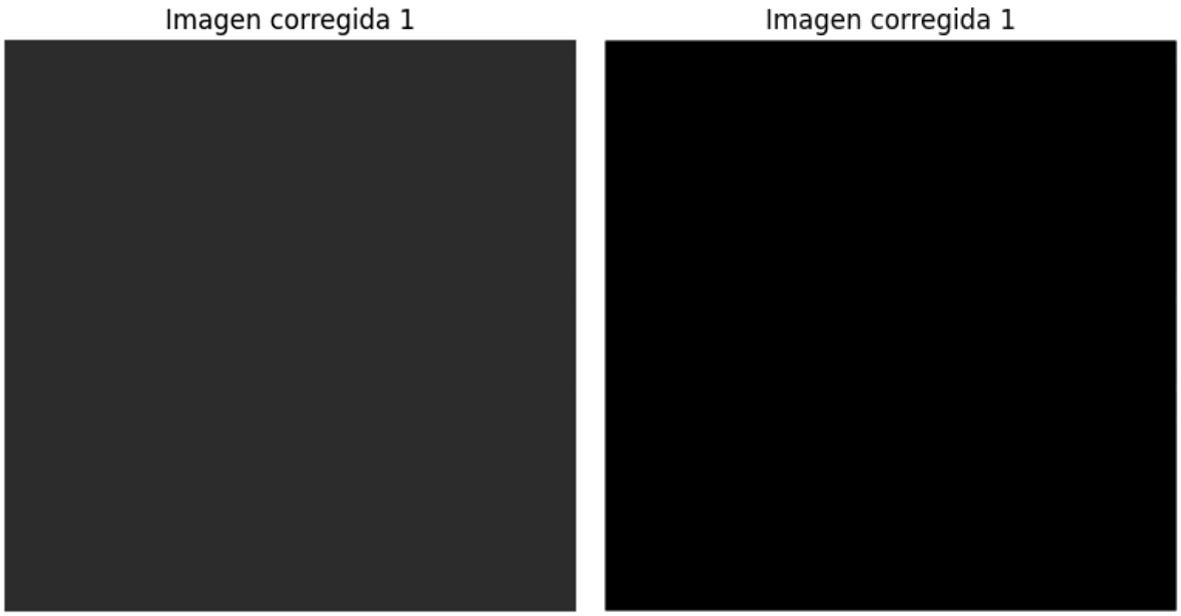

Figura 39: Imágenes predichas por el modelo de la prueba 21. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles

Tal y como se menciona anteriormente, para la prueba 22 se aumenta *clipnorm* hasta un valor de 1. Con ello se busca explorar si una técnica menos estricta de *clipping* puede ayudar al modelo a obtener mejores resultados. Los resultados obtenidos se muestran en la Figura 40, donde se observa que tienen cierto parecido a la prueba anterior, pero esta vez se alcanzan los mejores valores de entrenamiento de todas las pruebas realizadas, destacando valores de la pérdida de 0,027, del PSNR de 16,465 dB o del MS-SSIM de 0,655. Sin embargo, los valores de validación obtenidos en este caso son aun más reducidos que en la prueba anterior, siendo así la diferencia entre los resultados del entrenamiento y de la validación mayores. En el caso del SSIM en esta prueba no ha actuado como la prueba anterior, sufriendo una gran caída en la validación y terminando con un valor menor que el entrenamiento.

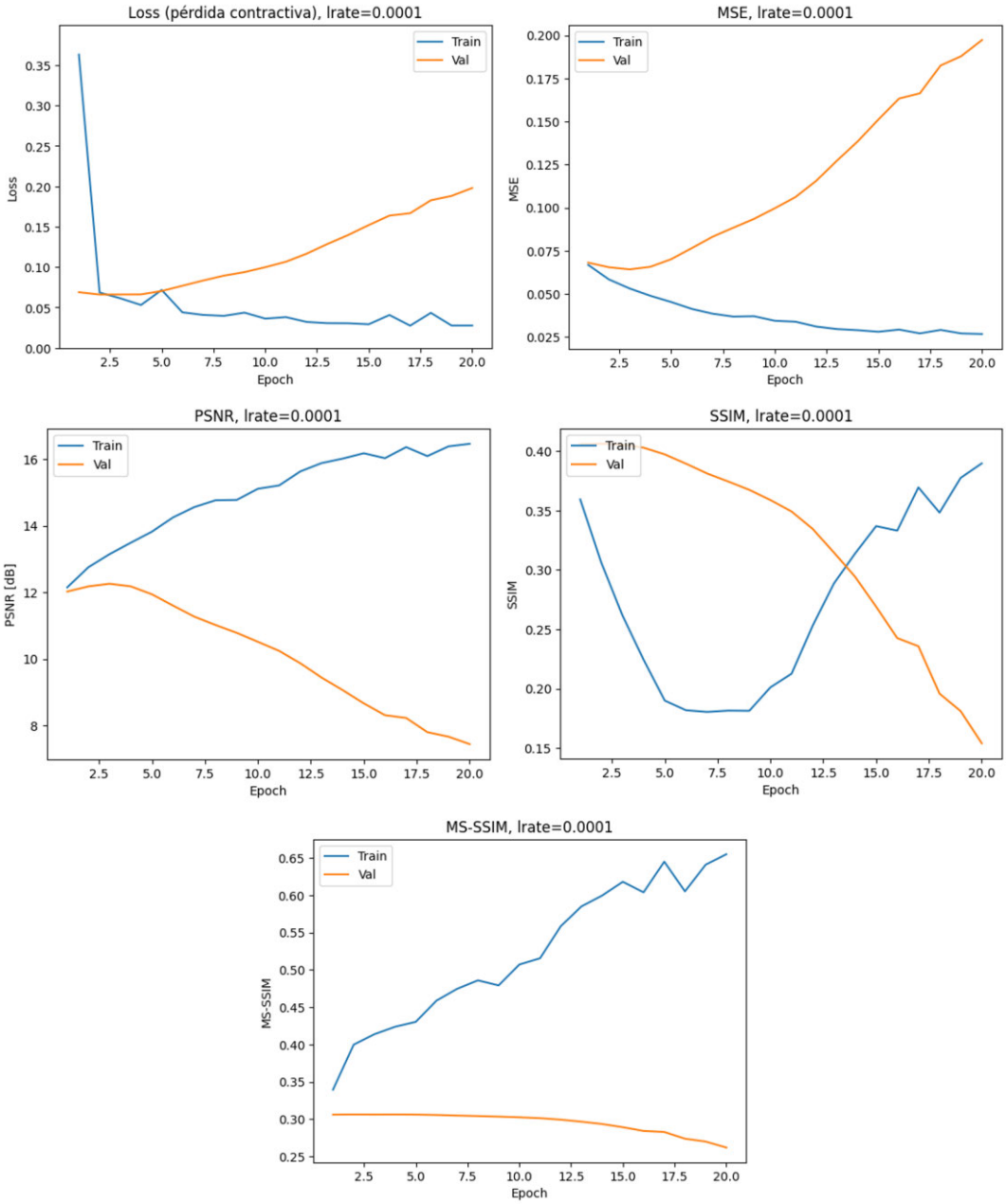

Figura 40: Resultados del entrenamiento de la prueba 22, con λ = 0,01, un learning rate de 10^{-4} y *clipnorm* de 1. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM

En esta prueba parece que se muestra otro caso más marcado de *overfitting*, donde el aumento de *clipnorm* no ha ayudado al modelo a mejorar el aprendizaje del comportamiento buscado. Esto también se destaca al observar los valores obtenidos al evaluar la red, mostrados en la Tabla 8, los cuales son los valores más bajos de todos los obtenidos al evaluar las distintas pruebas que no han sufrido subidas de la pérdida.

Tabla 8: Resultados de la evaluación de los valores del modelo de la prueba 22

Pérdida	**MSE**	**PSNR [dB]**	**SSIM**	**MS-SSIM**
0,170	0,170	8,216	0,249	0,293

De esta manera, las imágenes predichas, las cuales se pueden observar en la Figura 41, tampoco presentan ningún tipo de forma reconocible. El rango de los valores de los píxeles se encuentra esta vez entre 21 y 75, y por ello las imágenes se ven tan oscuras. Además, las imágenes deben de contener muchos valores alrededor de 21, ya que la imagen representada respecto a los valores mínimos y máximos se aprecia completamente negra.

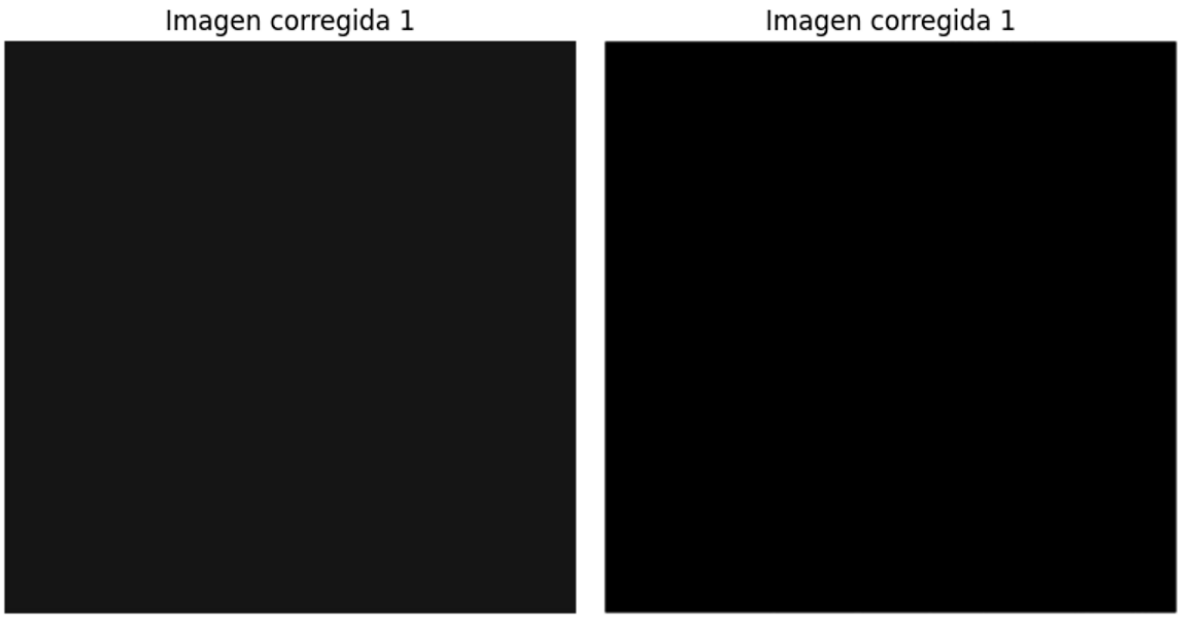

Figura 41: Imágenes predichas por el modelo de la prueba 22. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles

Con esta segunda prueba se confirma que el uso de *clipnorm* no resulta una técnica útil para este tipo de modelo, ya que se halla *overfitting* al ser utilizado. Para conseguir evitar el *overfitting* se plantea el uso de una técnica más restrictiva que pueda conseguir valores más cercanos a los buscados, ya que los resultados obtenidos hasta el momento no se acercan al objetivo propuesto del proyecto.

5.3.3. Tercera técnica: *clipvalue*

Como tercera y última técnica de *clipping* a estudiar se propone el uso del argumento *clipvalue* del optimizador *Adam*[34]. Este es parecido a *clipnorm* ya que trabaja con los gradientes de las capas, pero en este caso lo que realiza es limitar directamente el valor de los gradientes, no su norma. De esta manera, es más restrictivo que *clipnorm*, pero se elige porque podría actuar mejor y sin causar *overfitting*. Como se ha comprobado que técnicas muy restrictivas como *weight clipping* no han tenido un buen resultado, se utiliza con *clipvalue* un valor alto como máximo valor de los gradientes para que este no sea excesivamente restrictivo.

Con esta técnica se lleva a cabo la prueba 23, para la cual se utiliza un valor de lambda de 0,01, un valor de *learning rate* de 0,0001, y se establece el *clipvalue* con un valor de 5. Además, se utilizan un total de 15 épocas. Cabe destacar que se mantiene el valor de *weight_decay* en 10^{-5}.

En la Figura 42 se aprecian los resultados del entrenamiento de la prueba 23. Estos son algo similares a las pruebas anteriores, aunque presentan más irregularidades, especialmente en los valores de entrenamiento. En esta prueba los valores de validación vuelven a recibir peores resultados en comparación con los de entrenamiento, a excepción del SSIM, el cual percibe valores por debajo de la

media en los resultados de entrenamiento. En el caso de la pérdida, no aumenta tanto como ocurría en las pruebas anteriores, y el PSNR tampoco se reduce en exceso.

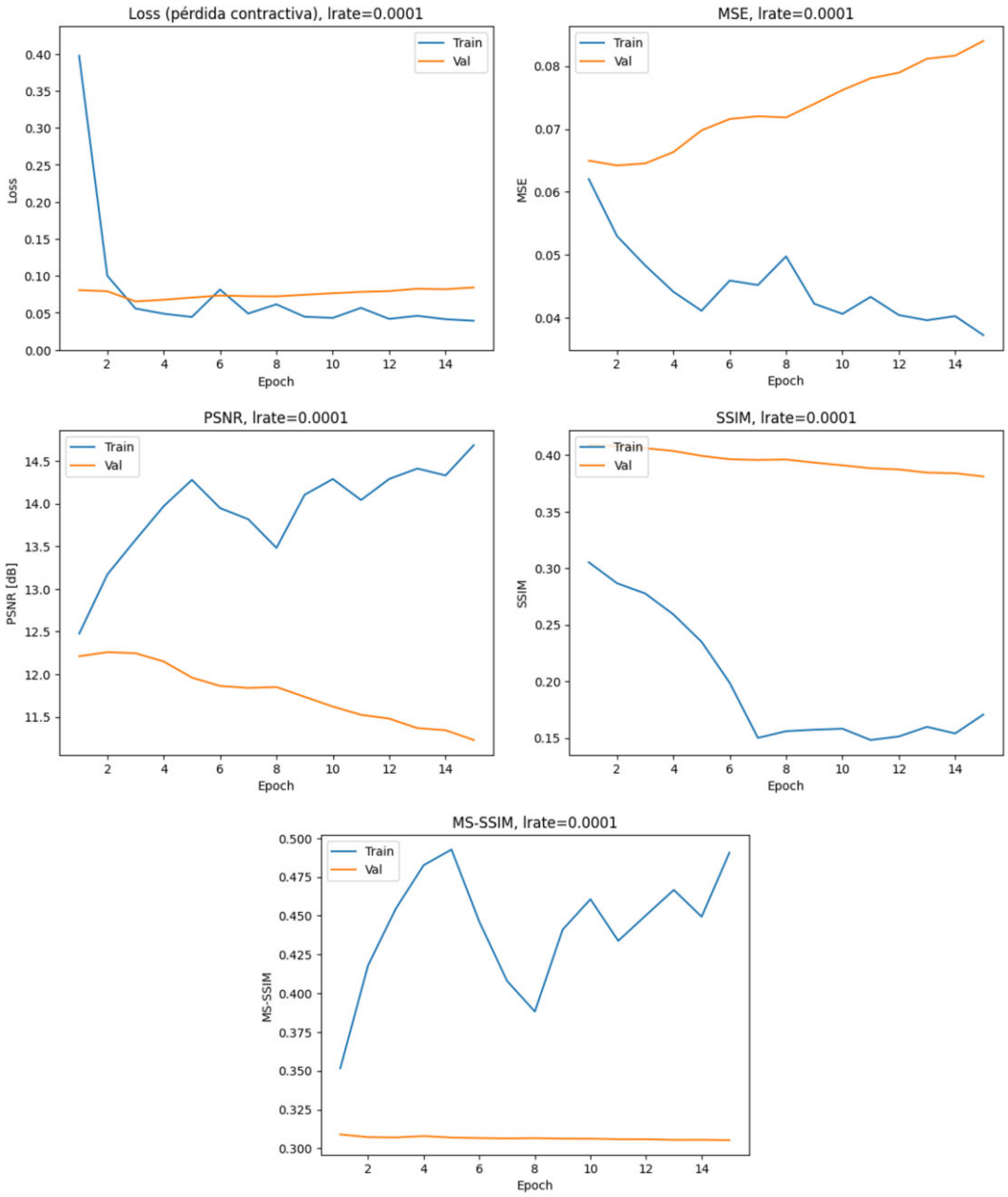

Figura 42: Resultados del entrenamiento de la prueba 23, con λ = 0,01, un learning rate de 10^{-4} y *clipvalue* de 5. Arriba: Valor de la pérdida y del MSE. Abajo: Valor de las métricas PSNR, SSIM y MS-SSIM

Estos resultados son algo mejores que las pruebas anteriores, donde el uso de *clipvalue* y la reducción del número de épocas han resultado beneficiosos. El número de épocas menor ayuda a la reducción del riesgo de *overfitting*, aunque aún se muestran algunos signos de un posible caso.

En la Tabla 9 se muestran los resultados de la evaluación del modelo. Cabe destacar que estos no son tan buenos como otras pruebas vistas anteriormente, pero sí que consiguen realizar una mejora en comparación con las pruebas 21 y 22.

Tabla 9: Resultados de la evaluación de los valores del modelo de la prueba 23

Pérdida	MSE	PSNR [dB]	SSIM	MS-SSIM
0,085	0,085	11,256	0,409	0,319

Se calculan las imágenes predichas y se muestra el resultado final en la Figura 43. Para este se ha obtenido un valor mínimo de los píxeles de 64, y un valor máximo de 90. Aunque los resultados de las métricas son mejores, no se consigue visualizar ninguna forma o estructura en la imagen representada.

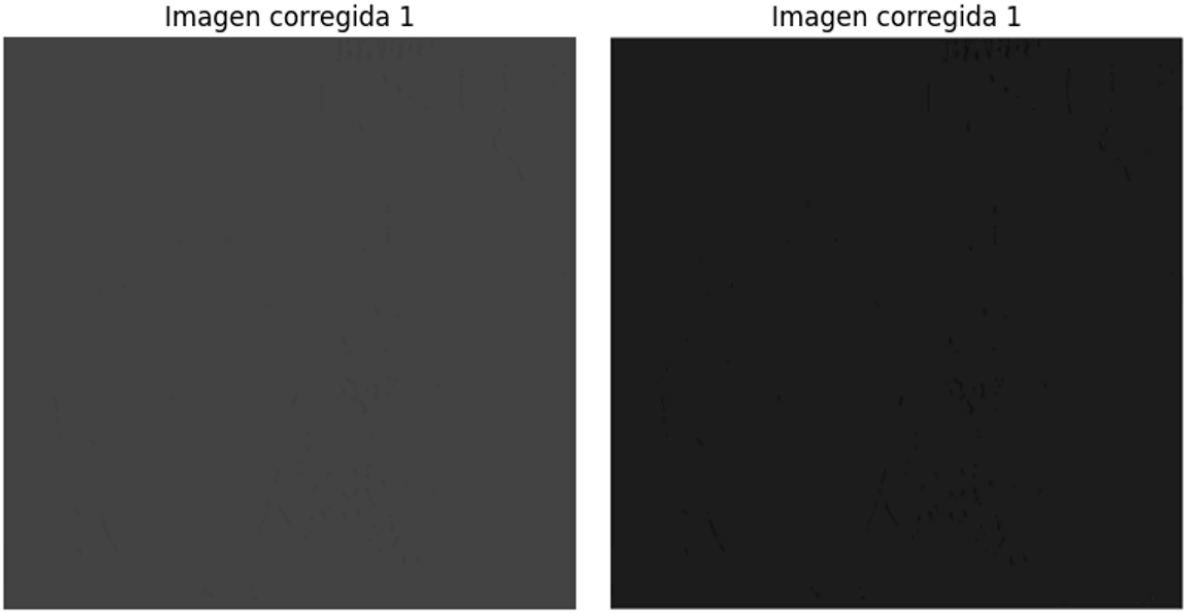

Figura 43: Imágenes predichas por el modelo de la prueba 22. Izquierda: Representación real de la imagen. Derecha: Representación en función de los valores mínimos y máximos de los píxeles

De esta manera, se demuestra que el uso de *clipvalue* no consigue alcanzar tampoco los objetivos buscados por el proyecto. Parece que con este se disminuye ligeramente el efecto del posible *overfitting*, aunque no se descarta otro tipo de afección. No se ha realizado una segunda prueba más restrictiva con esta técnica debido a la cercanía de los resultados respecto a las pruebas anteriores, con las que tampoco se ha alcanzado el objetivo, de manera que no se identifica esta técnica con la correcta para solucionar el problema encontrado. Además, se ha comprobado previamente que una técnica de *clipping* demasiado restrictiva puede conllevar a que el modelo no pueda aprender las características de las fotografías.

5.4. Discusión final

Las características de la red e hiperparámetros que mejores resultados aportan han sido los correspondientes a la prueba 19, debido a que sus imágenes predichas son las que más se asimilan a las imágenes de referencia. Sin embargo, las métricas obtenidas para otras pruebas también ofrecen resultados muy interesantes, aunque la imagen predicha no sea la esperada. Esto se debe a que si se comparan los resultados de las métricas con los ofrecidos en otros modelos del estado del arte ([2], [17]) ambos no difieren en gran cantidad.

Se puede comprobar esta diferencia en la Figura 44, que proviene de [17] y expone los resultados obtenidos por su modelo y por modelos anteriores, entre ellos [2], señalando en rojo los valores de las

métricas obtenidas con las imágenes de interés para este proyecto. Si se tienen en cuenta los resultados acumulados a lo largo de las pruebas de este proyecto, el PSNR obtenido en muchos de ellos se encuentra por encima de varios de los modelos expuestos, concretamente en la prueba 19 con un PSNR de 12,363 dB. Aunque el valor del SSIM no ha alcanzado los valores expuestos en la Figura 44, consiguiendo en la prueba 19 un valor de 0,440, esto se debe a que las imágenes predichas no reflejan aun la corrección de la exposición, por lo que este valor permanece algo inferior. Sin embargo, se puede identificar en las imágenes predichas de la prueba mencionada la estructura de la imagen *ground truth*, y por ello el valor del SSIM no es tan inferior como el obtenido en otras pruebas con resultados donde no se consigue que se refleje dicha estructura. Este efecto también se aprecia en la métrica MS-SSIM, que, aunque no ha sido estudiada en los modelos del estado del arte, se puede comprobar cómo un incremento en su valor se asocia con una mejora en la calidad de los resultados obtenidos.

Method	Expert A		Expert B		Expert C		Expert D		Expert E		Avg		PI
	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	
WVM [12]	12.355	0.624	13.147	0.656	12.748	0.645	14.059	0.669	15.207	0.690	13.503	0.657	2.342
LIME* [16, 17]	09.627	0.549	10.096	0.569	9.875	0.570	10.936	0.597	11.903	0.626	10.487	0.582	2.412
HDR CNN w/ PS [9]	14.804	0.651	15.622	0.689	15.348	0.670	16.583	0.685	18.022	0.703	16.076	0.680	2.248
DPED (iPhone) [21]	12.680	0.562	13.422	0.586	13.135	0.581	14.477	0.596	15.702	0.630	13.883	0.591	2.909
DPED (BlackBerry) [21]	15.170	0.621	16.193	0.691	15.781	0.642	17.042	0.677	18.035	0.678	16.444	0.662	2.518
DPE (HDR) [8]	14.399	0.572	15.219	0.573	15.091	0.593	15.692	0.581	16.640	0.626	15.408	0.589	2.417
DPE (S-FiveK) [8]	14.786	0.638	15.519	0.649	15.625	0.668	16.586	0.664	17.661	0.684	16.035	0.661	2.621
RetinexNet* [55]	10.149	0.570	10.880	0.586	10.471	0.595	11.498	0.613	12.295	0.635	11.059	0.600	2.933
Deep UPE* [51]	10.047	0.532	10.462	0.568	10.307	0.557	11.583	0.591	12.639	0.619	11.008	0.573	2.428
Zero-DCE [15]	10.116	0.503	10.767	0.502	10.395	0.514	11.471	0.522	12.354	0.557	11.021	0.519	2.774
Afifi et al. w/o L_{adv} [2]	18.976	0.743	19.767	0.731	19.980	0.768	18.966	0.716	19.056	0.727	19.349	0.737	2.189
Afifi et al. w/ L_{adv} [2]	18.874	0.738	19.569	0.718	19.788	0.760	18.823	0.705	18.936	0.719	19.198	0.728	2.183
Ours	20.475	0.862	21.833	0.891	22.438	0.901	20.127	0.874	20.062	0.881	20.987	0.881	2.158

Figura 44: Reproducción de *Table 1* de [17], resaltando con un rectángulo rojo los resultados de interés. Esta tabla contiene y amplía los resultados de [2]

6. Presupuesto

Debido a la naturaleza del proyecto, para calcular el presupuesto es necesario tener en cuenta pocos factores. Dentro de estos se incluyen el hardware y software utilizados para la realización del modelo, y la mano de obra.

El software sobre el que se desarrolla el código es Google Colaboratory, y aunque este ofrece una versión gratuita con ciertas horas de uso, esta no resulta suficiente para un desarrollo eficiente del modelo. Por ello, se adquiere la subscripción 'Colab Pro', con un precio de 11,19 euros mensuales [42]. El precio final depende de la cantidad de tiempo en el que se desarrolle este modelo, pero se utiliza un tiempo de referencia de 5 meses de uso.

Para el hardware se contempla el uso de un ordenador portátil personal. Como el modelo se ejecuta sobre Google Colab, el cual provee al usuario de la memoria RAM a utilizar, no es necesario que el ordenador personal cuente con especificaciones muy avanzadas, por lo que se establece un precio de referencia de 500 euros. Teniendo en cuenta que la vida útil promedio de un ordenador portátil personal se encuentra en 4 años, utilizando este a tiempo completo durante 5 meses, se establece el coste del uso del dispositivo en 52,10 euros.

Respecto a la mano de obra, el salario promedio en España de un ingeniero de telecomunicaciones ronda alrededor de 15,38 euros la hora [43]. Para realizar el cálculo se utiliza el tiempo de desarrollo del proyecto, estableciendo 360 horas de trabajo. Además, se aplica un suplemento del 30% debido a las cotizaciones sociales, quedando así en 20 euros por hora.

En la Tabla 10 se ofrece el presupuesto total de desarrollo del proyecto.

Tabla 10: Presupuesto del proyecto

Concepto	Unidades	Cantidad	Precio por unidad (euros)	Total (euros)
Subscripción Google Colab	Meses	5	11,19	55,95
Ordenador personal	Unidad	1	52,10	52,10
Sueldo ingeniero de telecomunicaciones	Horas	360	20,00	7200,00
TOTAL:				**7.308,05**

De esta manera, se contempla un presupuesto total de siete mil trescientos ocho con cinco euros, con una duración de trescientas sesenta horas.

7. Impacto del proyecto

Debido a la naturaleza del proyecto, su impacto e implicaciones se basan en la capacidad de mejora de la obtención de datos, en este caso en la mejora de imágenes ya existentes a través de la corrección de posibles problemas de exposición. Esto puede ayudar en diferentes ámbitos, de manera que el impacto a considerar es bastante amplio y podría afectar en actividades muy variadas.

Uno de los ámbitos más directos en ver los resultados puede ser el ámbito sanitario. En la actualidad ya hay muchos modelos de aprendizaje automático que se enfocan en este campo de estudio, y muchos de ellos rondan alrededor de la clasificación de imágenes, especialmente para la detección de tumores u obtención de diagnósticos. Todas estas tareas pueden beneficiarse si previo al uso de otros modelos se mejora la imagen inicial, ya que al mejorar la iluminación global puede ayudar a distinguir diferentes zonas de la imagen a procesar, de manera que esto facilita las tareas del siguiente modelo o procesado a utilizar sobre la imagen.

De esta misma manera, el proyecto puede tener implicaciones tanto sociales como tecnológicas. En un enfoque más personal, este proyecto puede ayudar a fotógrafos menos experimentados o con menor comprensión del uso de programas de edición avanzados a llevar a cabo correcciones de exposición, siendo estos errores muy comunes que les ocurren hasta a fotógrafos profesionales. Esto también se puede enfocar en un ámbito social más amplio, ya que el modelo final no requiere conocimientos previos, de manera que cualquier persona con la necesidad de realizar una corrección de la iluminación podría hacer uso de este. Tecnológicamente, el modelo supone una solución más sencilla en comparación a otros modelos similares ya existentes ([2], [17]), y puede ser utilizado como base para nuevos proyectos más avanzados, de manera que haya un mayor desarrollo en el procesado digital de la imagen.

Aunque se han ofrecido las más implicaciones más destacadas, el impacto del proyecto también se puede medir a través de los ODS (Objetivos de Desarrollo Sostenible)[44]. Estudiando estos se encuentran varios con los que puede servir de ayuda, presentados a continuación:

- Objetivo 3: Salud y bienestar. Tal y como se ha explicado anteriormente, este modelo puede suponer una ayuda en el procesado de imágenes médicas, ya que siendo utilizado como preprocesado de imágenes previo al uso de otros modelos ya existentes de clasificación o detección puede elevar la eficacia de estos, obteniendo así mejores resultados y diagnósticos más acertados.
- Objetivo 9: Industria, innovación y estructura. Este objetivo promueve la innovación tecnológica en diversos ámbitos, siendo uno de estos las tecnologías de la información y comunicación. El proyecto se encuentra dentro de este ámbito, concretamente en el marco de procesado avanzado de la señal. Además, el proyecto podría ser utilizado también dentro de procesos industriales si el proceso en cuestión tuviera la necesidad de utilizar cámaras, pudiendo solucionar errores en la captación de la imagen.
- Objetivo 14: Vida submarina. El proyecto podría ayudar a la vigilancia de mares y océanos, ya que el fondo marítimo es muy oscuro y al intentar estudiarlo podrían quedar imágenes subexpuestas. La meta del Objetivo 14 supone la conservación de océanos y mares, y este proyecto, aunque no sirva de ayuda para reducir la contaminación o acabar con la pesca no reglamentada, puede resultar útil para el estudio de la vida marina y del fondo de los océanos.

8. Conclusiones

8.1. Conclusiones

Tras la realización de este proyecto fin de carrera se ha logrado la creación de un modelo propio, a través del uso de una arquitectura tipo U-Net convolucional contractiva, y de la realización de funciones propias creadas dentro del contexto de este proyecto para el funcionamiento de la red, como la función de pérdida, la adaptación de las imágenes o la aplicación de técnicas de *clipping*. Además, constituye un modelo diferente al estado del arte, ya que la arquitectura utilizada es más sencilla y no se encuentra publicado un estudio igual.

Desde un principio se considera este proyecto de fin de carrera como ambicioso, al ser los modelos ya existentes mucho más desarrollados y llevados a cabo tras un gran periodo de tiempo y con grandes equipos de investigación. De esta manera, no se han alcanzado los mismos resultados que en este tipo de publicaciones. Sin embargo, en el presente proyecto se ha conseguido la creación de un modelo propio, que puede considerarse como la primera piedra en la construcción de un modelo más avanzado, capaz de llevar a cabo el objetivo final. Esta primera piedra adquiere una relevancia significativa en el contexto de un proyecto de fin de grado, dado que el proceso seguido no es sencillo ni sistemático, ya que no existen procedimientos preestablecidos para entrenar o ajustar modelos de este tipo.

De esta forma, con el proyecto se ha llevado una labor de investigación sobre el efecto de diferentes arquitecturas, técnicas, e hiperparámetros dentro de un modelo para la corrección de errores de exposición utilizando una estructura tipo U-Net convolucional contractiva. También se han estudiado los valores de las métricas obtenidas, observando el impacto del incremento y del decremento de sus valores en los resultados finales de las imágenes predichas.

8.2. Trabajos futuros

Tal y como ha sido mencionado, el objetivo final podría conseguirse a través de una continuación del proyecto actual. Esto se podría alcanzar a través de la refinación de su arquitectura y de la realización de un estudio profundo de los hiperparámetros utilizados.

Se sospecha que la clave para alcanzar el objetivo se basa en la estructura de capas del modelo. En el caso de la red de [17], se menciona que las capas convolucionales en el codificador y en el decodificador están estructuradas en cinco niveles, implementando también capas de normalización. No se especifica la cantidad de capas convolucionales en cada nivel, ni la cantidad de capas de normalización. También se hace mención del uso de la convolución traspuesta en el decodificador. De esta manera, una de las opciones que se explora para que el modelo sea capaz de generar las imágenes corregidas es la realización de pruebas con diferentes configuraciones de las capas. Respecto a estas configuraciones, se exploran entre otras posibilidades la incorporación de niveles adicionales con el mismo número de capas, la incorporación de niveles adicionales variando el número de capas, o la reducción del número de capas incluidas en cada nivel.

Además de la refinación y ajuste de la arquitectura del modelo, es importante llevar a cabo pruebas y realizar un estudio extenso de los hiperparámetros, para comprobar cuáles pueden conllevar un efecto positivo en los resultados. El objetivo que se persigue es la obtención de valores de las métricas competitivos, que puedan hacer frente a los modelos del estado del arte.

9. Referencias

[1] Xataka foto, "Triángulo de exposición en fotografía", 2019. [En línea]. Disponible en: https://www.xatakafoto.com/guias/triangulo-exposicion-explicado-graficamente. [Accedido: 28 de julio de 2024]

[2] M. Afifi, K. G. Derpanis, B. Ommer and M. S. Brown, "Learning Multi-Scale Photo Exposure Correction," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Nashville, TN, USA, 2021, pp. 9153-9163, doi: 10.1109/CVPR46437.2021.00904.

[3] Wikipedia, "Valor de exposición". [En línea]. Disponible en: https://es.wikipedia.org/wiki/Valor_de_exposici%C3%B3n. [Accedido: 29 de julio de 2024]

[4] G. Ciaburro and B. Venkateswaran, *Neural Networks with R : Smart models using CNN, RNN, deep learning, and artificial intelligence principles.*, 1st ed. 2017.

[5] R. Han, "Convergence Analysis of Different Categories of Gradient Descent Algorithms," *2023 3rd International Conference on Computer Science, Electronic Information Engineering and Intelligent Control Technology (CEI)*, Wuhan, China, 2023, pp. 285-289, doi: 10.1109/CEI60616.2023.10527829.

[6] D. Dai, "An Introduction of CNN: Models and Training on Neural Network Models," *2021 International Conference on Big Data, Artificial Intelligence and Risk Management (ICBAR)*, Shanghai, China, 2021, pp. 135-138, doi: 10.1109/ICBAR55169.2021.00037.

[7] AnimatedAI, "Convolution". [En línea]. Disponible en: https://animatedai.github.io/. [Accedido: 1 de agosto de 2024]

[8] E. Ebermam and R. Krohling, "A Comprehensive Introduction to Neural Convolutional Networks: A Case Study for Character Recognition," Sistemas de Informação (Macaé), vol. 1, no. 22, pp. 49–59, 2018.

[9] Y. Zhang, "A Better Autoencoder for Image: Convolutional Autoencoder," 2018. [En línea]. Disponible: http://users.cecs.anu.edu.au/~Tom.Gedeon/conf/ABCs2018/paper/ABCs2018_paper_58.pdf. [Accedido: 1 de agosto de 2024]

[10] B. Li, K. Xu, D. Feng, H. Mi, H. Wang and J. Zhu, "Denoising Convolutional Autoencoder Based B-mode Ultrasound Tongue Image Feature Extraction," *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Brighton, UK, 2019, pp. 7130-7134, doi: 10.1109/ICASSP.2019.8682806.

[11] M. Naderan and Y. Zaychenko, "Convolutional Autoencoder Application for Breast Cancer Classification," *2020 IEEE 2nd International Conference on System Analysis & Intelligent Computing (SAIC)*, Kyiv, Ukraine, 2020, pp. 1-4, doi: 10.1109/SAIC51296.2020.9239139.

[12] A. Tursynova and B. Omarov, "3D U-Net for brain stroke lesion segmentation on ISLES 2018 dataset," *2021 16th International Conference on Electronics Computer and Computation (ICECCO)*, Kaskelen, Kazakhstan, 2021, pp. 1-4, doi: 10.1109/ICECCO53203.2021.9663825.

[13] GitHub, "Project page of the paper “Learning Multi-Scale Photo Exposure Correction”". [En línea]. Disponible en: https://github.com/mahmoudnafifi/Exposure_Correction?tab=readme-ov-file. [Accedido: 5 de agosto de 2024]

[14] A. Horé and D. Ziou, "Image Quality Metrics: PSNR vs. SSIM," *2010 20th International Conference on Pattern Recognition*, Istanbul, Turkey, 2010, pp. 2366-2369, doi: 10.1109/ICPR.2010.579.

[15] R. Liu, L. Ma, Y. Zhang, X. Fan and Z. Luo, "Underexposed Image Correction via Hybrid Priors Navigated Deep Propagation," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 8, pp. 3425-3436, Aug. 2022, doi: 10.1109/TNNLS.2021.3052903.

[16] L. Zhao, S. -P. Lu, T. Chen, Z. Yang and A. Shamir, "Deep Symmetric Network for Underexposed Image Enhancement with Recurrent Attentional Learning," *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, Montreal, QC, Canada, 2021, pp. 12055-12064, doi: 10.1109/ICCV48922.2021.01186

[17] F. I. Eyiokur, D. Yaman, H. K. Ekenel and A. Waibel, "Exposure Correction Model to Enhance Image Quality," *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, New Orleans, LA, USA, 2022, pp. 675-685, doi: 10.1109/CVPRW56347.2022.00083.

[18] "TensorFlow". [En línea]. Disponible en: https://www.tensorflow.org/?hl=es-419. [Accedido: 7 de agosto de 2024]

[19] "Keras: Deep Learning for humans". [En línea]. Disponible en: https://keras.io/. [Accedido: 7 de agosto de 2024]

[20] "scikit-image: Image processing in Python". [En línea]. Disponible en: https://scikit-image.org/. [Accedido: 7 de agosto de 2024]

[21] "Matplotlib: Visualization with Python". [En línea]. Disponible en: https://matplotlib.org/. [Accedido: 7 de agosto de 2024]

[22] W3Schools, "Python os Module". [En línea]. Disponible en: https://www.w3schools.com/python/module_os.asp?ref=escape.tech. [Accedido: 7 de agosto de 2024]

[23] W3Schools, "Python JSON". [En línea]. Disponible en: https://www.w3schools.com/python/python_json.asp. [Accedido: 7 de agosto de 2024]

[24] "NumPy". [En línea]. Disponible en: https://numpy.org/. [Accedido: 7 de agosto de 2024]

[25] Keras, "Conv2D layer". [En línea]. Disponible en: https://keras.io/api/layers/convolution_layers/convolution2d/. [Accedido: 26 de agosto de 2024]

[26] Keras, "MaxPooling2D layer". [En línea]. Disponible en: https://keras.io/api/layers/pooling_layers/max_pooling2d/. [Accedido: 26 de agosto de 2024]

[27] Keras, "UpSampling2D layer". [En línea]. Disponible en: https://keras.io/api/layers/reshaping_layers/up_sampling2d/. [Accedido: 26 de agosto de 2024]

[28] Keras, "Concatenate layer". [En línea]. Disponible en: https://keras.io/api/layers/merging_layers/concatenate/. [Accedido: 26 de agosto de 2024]

[29] Keras, "Dropout layer". [En línea]. Disponible en: https://keras.io/api/layers/regularization_layers/dropout/. [Accedido: 26 de agosto de 2024]

[30] MIT, "Adobe FiveK dataset". [En línea]. Disponible en: https://data.csail.mit.edu/graphics/fivek/. [Accedido: 6 de agosto de 2024]

[31] Keras, "Rescaling layer". [En línea]. Disponible en: https://keras.io/api/layers/preprocessing_layers/image_preprocessing/rescaling/. [Accedido: 7 de agosto de 2024]

[32] Tensorflow, "tf.keras.preprocessing.image_dataset_from_directory". [En línea]. Disponible en: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image_dataset_from_directory. [Accedido: 7 de agosto de 2024]

[33] Tensorflow, "tf.data.Dataset". [En línea]. Disponible en: https://www.tensorflow.org/api_docs/python/tf/data/Dataset. [Accedido: 7 de agosto de 2024]

[34] Tensorflow, "tf.keras.optimizers.Adam". [En línea]. Disponible en: https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam. [Accedido: 23 de agosto de 2024]

[35] Keras, "Model training APIs". [En línea]. Disponible en: https://keras.io/api/models/model_training_apis/. [Accedido: 23 de agosto de 2024]

[36] Keras, "Regression losses". [En línea]. Disponible en: https://keras.io/api/losses/regression_losses/. [Accedido: 24 de agosto de 2024]

[37] Keras, "Layers API". [En línea]. Disponible en: https://keras.io/api/layers/. [Accedido: 23 de agosto de 2024]

[38] Tensorflow, "tf.GradientTape". [En línea]. Disponible en: https://www.tensorflow.org/api_docs/python/tf/GradientTape. [Accedido: 24 de agosto de 2024]

[39] Keras, "EarlyStopping". [En línea]. Disponible en: https://keras.io/api/callbacks/early_stopping/. [Accedido: 24 de agosto de 2024]

[40] Matplotlib, "matplotlib.pyplot.imshow". [En línea]. Disponible en: https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.imshow.html. [Accedido: 28 de agosto de 2024]

[41] Keras, "Callbacks API". [En línea]. Disponible en: https://keras.io/api/callbacks/. [Accedido: 30 de agosto de 2024]

[42] Google LLC, "Google Colab". [En línea]. Disponible en: https://colab.research.google.com/. [Accedido: 13 de julio de 2024]

[43] Talent, "Salario para Ingeniero Telecomunicaciones en España". [En línea]. Disponible en: https://es.talent.com/salary?job=ingeniero+telecomunicaciones. [Accedido: 13 de julio de 2024]

[44] ONU, "Objetivos y metas de desarrollo sostenible". [En línea]. Disponible en: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/. [Accedido: 25 de julio de 2024]

Anexos

A.1 Función de adaptación de los datos de salida

Esta función adapta los datos de *ground truth*, quintuplicando las imágenes para poder enlazar una imagen de entrada con una de su salida esperada e introducir estas al modelo.

```
# Directorios de los archivos para buscar las imágenes originales y copiarlas
# en un nuevo directorio
y_orig_dir = '/content/drive/MyDrive/Colab Notebooks/PFG/DATASETS/val/GT_IMAGES_ORIGINAL'
save_dir = '/content/drive/MyDrive/Colab Notebooks/PFG/DATASETS/val/GT_IMAGES_QUINTUPLICADO'

# Se itera dentro del directorio para obtener los archivos de uno en uno
for archivo in os.listdir(y_orig_dir): # itera y obtiene el nombre del archivo
    ima = imread(os.path.join(y_orig_dir, archivo)) # Se lee la imagen GT iterada

    # Se separa el nombre del archivo de su extensión (.jpg) y se crea una lista
    # que contiene ambos strings, usando el . como lugar de separación
    nombre = archivo.split(".") # El nombre sin la extensión se guarda en nombre[0]
    for i in range(5):
        imsave(save_dir + "/" + nombre[0] + "_" + str(i+1) + ".jpg", ima)
```

A.2 Función de pérdida

Se muestra la clase ContractiveLossLayer, que contiene el cálculo de la función de pérdida.

```
class ContractiveLossLayer(Layer):
    def __init__(self, lam=0.01, **kwargs):
        super(ContractiveLossLayer, self).__init__(**kwargs)
        self.lam = lam

    def call(self, y_true, y_pred):
        mse = tf.keras.losses.MeanSquaredError()
        reconstruction_loss = mse(y_true, y_pred)

        # Calcula los grandientes con GradientTape(), que "graba" las operaciones
        # que ocurren y con ello calcula los gradientes
        with tf.GradientTape() as tape:
            tape.watch(y_pred) # vigila la variable y_pred (salida del modelo)
            reconstruction = model(y_pred) # calcula la reconstrucción con la anterior imagen predicha
        gradients = tape.gradient(reconstruction, y_pred) # calcula los gradientes

        # Calcula la norma de Frobenius (sin la raíz y así no hay que elevar al cuadrado)
        frobenius_norm = tf.reduce_sum(tf.square(gradients), axis=[1,2,3])

        # Calcula la pérdida contractiva
        contractive_loss = self.lam * tf.reduce_sum(frobenius_norm)

        return reconstruction_loss + contractive_loss
```

A.3 Clase *WeightClippingCallback*

Se muestra la clase *WeightClippingCallback*, creada para la realización de la técnica de w*eight clipping*.

```
class WeightClippingCallback(tf.keras.callbacks.Callback):
    def __init__(self, min_value, max_value):
        super(WeightClippingCallback, self).__init__()
        self.min_value = min_value
        self.max_value = max_value

    def on_train_batch_end(self, batch, logs=None):
        for layer in self.model.layers:
            weights = layer.get_weights()
            clipped_weights = [tf.clip_by_value(w, self.min_value, self.max_value) for w in weights]
            layer.set_weights(clipped_weights)
```