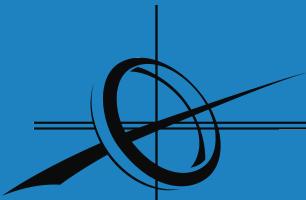




POLITÉCNICA



Universidad
Politécnica
de Madrid

**ETSI SISTEMAS
INFORMÁTICOS**

Generación de cuadros impresionistas mediante Redes Neuronales

Proyecto Fin de Grado

Grado en Ingeniería de Computadores

Autor:

Victor Manuel Domínguez Rivas

Tutores:

Luis Fernando de Mingo

Nuria Gómez Blas

Octubre 2020

UNIVERSIDAD POLITÉCNICA DE MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
SISTEMAS INFORMÁTICOS



Generación de cuadros impresionistas mediante Redes Neuronales

Proyecto Fin de Grado

Grado en Ingeniería de Computadores

Curso académico 2019-2020

Autor:
Victor Manuel Domínguez Rivas

Tutores:
Luis Fernando de Mingo
Nuria Gómez Blas

*A mis tutores por su apoyo en esta ardua tarea
que ha supuesto tanto para mí.*

*A mis amigos de la universidad, especialmente
a Juan Luis y a Rocío por su apoyo incondicional
y por hacerme tan feliz.*

*A mi familia por confiar en mí
en momentos tan complicados.*

*A mis amigos de mi pueblo, especialmente
a Javi, Polo, Miriam y Ana
por estar siempre a mi lado
a pesar de la distancia.*

*A Clara y a Sara por ser tan leales
y tan enriquecedoras conmigo.*

*A Paula, por redescubrirme el arte
y darme la idea de este proyecto.*

*Y por último a los profesores
que me dieron la motivación
y el apoyo que necesitaba.*

Resumen

Una parte intrínseca del ser humano es la habilidad de crear. El desarrollo de las civilizaciones humanas ha potenciado la creatividad humana con nuevas formas de expresión artística y tendencias rompedoras, a la vez que se convertía un elemento crucial para entender la sociedad del momento. Una de las corrientes artísticas más destacadas en la historia fue el Impresionismo pictórico, que a través de la luz y el color plasmaban en sus cuadros la fugacidad y la incertidumbre del momento, lo continuo y lo indefinido.

Por otra parte, en el último siglo se ha producido una explosión de conocimiento y de avances tecnológicos. La esperanza de vida española aumentó en 40 años durante ese periodo, aparecieron elementos transgresores como la televisión y el computador, se popularizaron masivamente inventos ya existentes como el cine, el teléfono y el automóvil... transformando completamente nuestra sociedad y nuestro estilo de vida.

Uno de estos avances, el computador, ha cobrado aún más importancia en lo que llevamos de siglo XXI. Los increíbles avances en potencia de cálculo, dispositivos, comunicaciones... han sembrado el terreno para que Internet y los ordenadores sean parte intrínseca de nuestras vidas. En este contexto, la Inteligencia Artificial toma cada vez más protagonismo en los últimos tiempos. Por primera vez en la historia de la Humanidad, hemos sido capaces de crear máquinas que toman decisiones por sí solas y con un impacto real y profundo en nuestras vidas. Por tanto suena razonable que nos hagamos la siguiente pregunta: ¿pueden las máquinas crear algo tan humano como es el Arte?

Este Trabajo Fin de Grado pretende acercar al lector a la Inteligencia Artificial a través de una perspectiva artística. Para ello se han tomado los cuadros de tres grandes pintores impresionistas y postimpresionistas como son Claude Monet, Vicent Van Gogh y Paul Cézanne; con el objetivo de implementar un sistema basado en Redes Neuronales que transforme fotografías en cuadros que el propio lector pueda utilizar en su computador.

Asimismo este documento espera concienciar de forma amena y accesible al lector de los avances producidos en la materia, con el objetivo que adquiera una mirada crítica sobre los sistemas de Inteligencia Artificial que utiliza en su día a día; debido a las cuestiones éticas, morales y sociales que plantean en nuestra sociedad.

Palabras clave: Impresionismo, Inteligencia Artificial, Redes Neuronales, Computación en la nube, Transferencia de estilo

Abstract

An intrinsic part of being human is the ability to create. The development of human civilizations has enhanced human creativity with new forms of artistic expression and revolutionary trends; while it has become a crucial element to understand the society of the moment. One of the most prominent artistic movements in history was pictorial Impressionism, using light and color in order to capture the fleetingness and uncertainty of the moment, the continuous and the indefinite in their paintings.

Furthermore, there has been an explosion of knowledge and technological advances during 20th century. Life expectancy increased by 40 years (in Spain) since 1920, transgressive elements such as television and computer appeared on our lives, inventions like cinema, telephone and automobile were massively popularized... transforming completely our society and lifestyle.

One of these advances, the computer, has become even more important in the 21st century. The incredible advances in computing power, devices, communications... facilitated that Internet and computers became an intrinsic part of our lives. In this context, Artificial Intelligence takes more and more prominence nowadays. For the first time in human history, we have been able to create machines that make decisions for themselves and with a real and profound impact on our lives. Therefore we could ask ourselves the following question: can machines create something so human as Art?

This Final Degree Project aims to bring the reader closer to Artificial Intelligence through an artistic perspective. For this objective, the paintings of three great impressionist and post-impressionist artists have been taken, such as Claude Monet, Vicent Van Gogh and Paul Cézanne; aiming to develop a system based on Neural Networks that transforms photographs into pictures that the reader himself can use on his computer.

Finally, the author hopes to make the reader aware about the advances produced in the field in a nicely and accessible way. Also, this document aims of acquiring a critical sense about the Artificial Intelligence systems that the reader uses in his everyday life due to the ethical, moral and social issues that these systems raise in our society.

Keywords: Impressionism, Artificial Intelligence, Neural Networks, Cloud Computing, Style Transfer

Índice

Agradecimientos	I
Resumen	II
Abstract	III
1. Introducción	1
1.1. Contexto	1
1.2. Objetivos	2
1.2.1. Objetivos secundarios	2
1.3. Estructura del documento	3
2. Estado del arte	4
2.1. Impresionismo Pictórico	4
2.1.1. Origen del nombre	4
2.1.2. Contexto artístico e histórico	5
2.1.3. Características artísticas	6
2.1.4. Monet	7
2.1.5. Van Gogh	8
2.1.6. Cézanne	10
2.2. Inteligencia Artificial y Redes Neuronales	11
2.2.1. ¿Qué es la Inteligencia Artificial?	11
2.2.2. Fundamentos de Machine Learning	12
2.2.3. ¿Por qué usar Machine Learning?	13
2.2.4. ¿Qué son las Redes Neuronales?	15
2.2.5. ¿Por qué usar Redes Neuronales?	17
2.2.6. Redes Neuronales Convolucionales	18
2.2.7. Redes Generativas Antagónicas	20
2.2.8. Transferencia de estilo y Cycle GAN	23
3. Desarrollo del proyecto	25
3.1. Características técnicas de los modelos a implementar	25
3.1.1. Cycle GAN	25
3.1.2. Métricas de la CycleGAN	27
3.1.3. EnhanceNet	28
3.2. Datos utilizados para elaborar el sistema	30
3.3. Tecnologías utilizadas	31
3.3.1. Python	31
3.3.2. TensorFlow	32
3.3.3. Tensorboard	32
3.3.4. Keras	33
3.3.5. CUDA	34
3.3.6. Google Cloud Plataform	34
3.3.7. Flask	34
3.3.8. Git y GitHub	35
3.4. Diseño general del sistema	36
3.4.1. Sistema principal	36
3.4.2. Sistema ampliador de resolución	40
3.5. Guía de uso	41

3.5.1. Instalación de dependencias	41
3.5.2. Creación de cuadros	42
3.5.3. Creación de modelos personalizados	43
4. Resultados	45
4.1. Resultados obtenidos	45
4.1.1. Paso de fotografías a cuadros	45
4.1.2. Paso de cuadros a fotografías	52
4.2. Objetivos logrados	56
4.3. Problemas encontrados	57
5. Conclusiones y trabajos futuros	58
5.1. Conclusiones	58
5.2. Impacto social y medioambiental	59
5.3. Lineas futuras	61
Bibliografía	62
Anexos	66
A. Recursos hardware y software utilizados	67
A.1. Hardware	67
A.1.1. Equipo personal del autor	67
A.1.2. Equipo contratado en Google Cloud Platform	67
A.2. Software	67
B. Código fuente del proyecto	69
B.1. Sistema principal	69
B.1.1. ampliar_imagen.py	69
B.1.2. capas_extra.py	70
B.1.3. cargador_imagenes.py	70
B.1.4. cycleGAN.py	73
B.1.5. lanzadera_entreno.py	84
B.1.6. lanzadera_produccion.py	85
B.1.7. procesado_imagenes.py	86
B.1.8. singleton.py	88
B.1.9. utilidades.py	89
B.1.10. requirements.txt	94
B.2. Archivos de configuración del sistema principal	96
B.2.1. configuracion_128.json	96
B.2.2. configuracion_256.json	97
B.2.3. configuracion_512.json	97
B.3. Sistema ampliador	99
B.3.1. controlador_modelo.py	99
B.3.2. modelo.py	99
B.3.3. principal.py	101
B.3.4. procesado.py	101
B.3.5. utils.py	103
B.3.6. requirements.txt	103

Índice de tablas

2.1. Comparación entre algunas de las funciones de activación más populares	16
---	----

Índice de figuras

1.1.	Alan Turing con 16 años	1
1.2.	Almuerzo sobre la hierba, de Édouard Manet	1
2.1.	Impresión, Amanecer, de Claude Monet	4
2.2.	Las espigadoras, de Jean-François Millet	5
2.3.	Grupo familiar ante un paisaje, de Frans Hals	5
2.4.	El Sena en Bercy, de Paul Cézanne	6
2.5.	La terraza de Sainte-Adresse, de Claude Monet	7
2.6.	Retrato de Claude Monet, por Pierre-Auguste Renoir.	7
2.7.	Mujer con sombrilla mirando a la izquierda, de Claude Monet	8
2.8.	Autorretrato, de Van Gogh	9
2.9.	La noche estrellada, de Van Gogh	9
2.10.	Autorretrato con sombrero arrugado, de Paul Cézanne	10
2.11.	Los jugadores de cartas, de Paul Cézanne	10
2.12.	Ejemplo de clustering.	12
2.13.	Ciclo de vida de una aplicación de Machine Learning.	14
2.14.	Ejemplo de underfitting, ajuste óptimo y overfitting.	14
2.15.	Representación gráfica de la función de una neurona artificial.	15
2.16.	Perceptrón y Red Neuronal Profunda	17
2.17.	Arquitectura de una RNC	19
2.18.	Operación de convolución	19
2.19.	Operación de reducción	20
2.20.	Fotos de rostros humanos creados por una red GAN	20
2.21.	Esquema de una red GAN	21
2.22.	Montaje de Zelda y La noche estrellada	24
2.23.	Comparativa entre pix2pix y Cycle GAN	24
3.1.	Arquitectura U-net	25
3.2.	Arquitectura resnet	26
3.3.	Bloque resnet	26
3.4.	Medidas de validez y reconstrucción	28
3.5.	Importancia del error identidad	28
3.6.	Ejemplo de EnhanceNet	29
3.7.	Arquitectura de EnhanceNet	29
3.8.	Vista previa del dataset <i>cezanne2photo</i>	30
3.9.	Datasets creados por el equipo de Cycle GAN	31
3.10.	Ejemplo de las gráficas proporcionadas por Tensorboard	33
3.11.	Ejemplo de la visualización de fotos mediante Tensorboard	33
3.12.	Ejemplo de visualización de <i>logs</i> de TensorBoard en el servicio de almacenamiento de Google Cloud Plataform	35
3.13.	Visualización de la descripción de la máquina virtual utilizada en este TFG alojada en Google Cloud Plataform	35
3.14.	Muestra de logs del sistema	36
3.15.	Ejemplo de archivo de configuración json	38
3.16.	Requisitos de versiones de nVidia de TensorFlow	42
3.17.	Ejemplo de contenido input	43

3.18. Ejemplo de cuadros generados	43
3.19. Parámetros para un nuevo dataset	44
3.20. Parámetro url para un nuevo dataset	44
4.1. Cuadro de Cézanne generado en el desierto	45
4.2. Cuadro de Cézanne generado en la noche	45
4.3. Cuadro de Cézanne generado en una orilla	46
4.4. Cuadro de Cézanne generado de unos edificios	46
4.5. Cuadro de Monet generado a partir de un edificio japonés de noche	46
4.6. Cuadro de Monet generado a partir de un puente	47
4.7. Cuadro de Monet generado a partir de edificaciones en el agua .	47
4.8. Cuadro de Monet generado a partir de un puente de madera . .	47
4.9. Cuadro de Van Gogh generado a partir de un campo con bolas .	48
4.10. Cuadro de Van Gogh generado a partir de una casa en el mar .	48
4.11. Cuadro de Van Gogh generado a partir de una montaña	48
4.12. Cuadro de Van Gogh a partir de un ferrocarril en el agua	48
4.13. Cuadro de Cézanne generado en el arco del triunfo de Barcelona	49
4.14. Cuadro de Cézanne generado a partir de una foto en una playa del autor	49
4.15. Cuadro de Cézanne generado de una foto de un parque de Madrid	49
4.16. Cuadro de Cézanne generado de una foto en el puerto de Honduras (Cáceres)	50
4.17. Cuadro de Cézanne generado de una Hyosung GTR 650	50
4.18. Cuadro de Monet generado a partir de una iglesia costera portuguesa	50
4.19. Cuadro de Monet generado en el parque Warner	51
4.20. Cuadro de Monet generado en el Palacio de Cristal de Madrid .	51
4.21. Cuadro de Monet generado en el centro de salud de Montehermoso (Cáceres)	51
4.22. Cuadro de Van Gogh generado en la estación de Chamartín (Madrid)	52
4.23. Cuadro de Van Gogh generado a partir de una foto del autor en Hervás (Cáceres)	52
4.24. Cuadro de Van Gogh generado a partir de una foto del autor en la Dehesa Boyal de Montehermoso (Cáceres)	52
4.25. Cuadro de Van Gogh a partir de una Hyosung GTR 650 estacionada en el pantano de Gabriel y Galán (Cáceres)	53
4.26. Foto basada en el cuadro de Cézanne <i>Marronniers et ferme au Jas de Bouffan</i>	53
4.27. Foto basada en el cuadro de Cézanne <i>Landscape. Study after Nature</i>	53
4.28. Foto basada en el cuadro de Cézanne <i>Homme à la pipe</i>	54
4.29. Foto basada en el cuadro de Monet <i>Callejón cerca de Pourville</i> .	54
4.30. Foto basada en el cuadro de Monet <i>The Promenade at Argenteuil</i>	54
4.31. Foto basada en el cuadro de Monet <i>El Verano</i>	55
4.32. Foto basada en el cuadro de Van Gogh <i>Un par de zuecos de cuero</i>	55
4.33. Foto basada en el cuadro de Van Gogh <i>Orchard Bordered by Cypresse</i>	55
4.34. Foto basada en el cuadro de Van Gogh <i>Jarrón con catorce girasoles</i>	56

Capítulo 1

Introducción

1.1. Contexto

En 1947, Alan Turing en su conferencia ante la London Mathematical Society [1] aportaba su visión a la vieja pregunta: ¿pueden pensar las máquinas? y proponía el famoso Test de Turing, objeto de inmensa influencia y debate en el campo de la Inteligencia Artificial. Mucho ha transcurrido desde entonces, con el debate sobre la capacidad de las máquinas transcendiendo los límites del mundo académico para impregnar el imaginario colectivo, con películas como Terminator (1984).



Figura 1.1: Alan Turing con 16 años. Imagen obtenida de [2].

Por otra parte, una de las formas de creatividad humana por excelencia es la pintura. Una de sus corrientes, el Impresionismo, buscaba reaccionar contra la pintura convencional plasmando sensaciones y la fugacidad del instante [3].



Figura 1.2: Almuerzo sobre la hierba, de Édouard Manet (Musée d'Orsay, París). Extraído de [4].

Asimismo, el Impresionismo fue un movimiento muy influyente en su apogeo (segunda mitad del siglo XIX) debido a que es el punto de unión entre dos fenómenos: la captación de la realidad por el artista mediante la pintura, iniciada en los albores del siglo XV; y la génesis del Arte Contemporáneo, en la que las *emociones* adquieren más peso en la técnica del artista, por lo que no cabe duda que fue un movimiento extremadamente creativo.

De vuelta al siglo XXI, en el mundo de la Inteligencia Artificial se han producido enormes avances en los últimos años: tenemos máquinas que pueden vencer a jugadores profesionales en juegos de gran tradición y prestigio como Go (AlphaGo versus Lee Sedol) [5], avanzados sistemas de reconocimiento facial implantados masivamente (Proyecto Skynet) [6], investigaciones sobre la telemedicina y el telediagnóstico en nuestra propia escuela [7] entre otros muchos, lo que ha originado una auténtica tormenta de controversia sobre las capacidades de la Inteligencia Artificial y de sus implicaciones éticas y morales.

Todo esto da pie a plantearse la pregunta: ¿puede crear arte una máquina? [8] [9], y de ser así ¿podría realizar algo tan creativo como cuadros impresionistas?

1.2. Objetivos

El objetivo principal de este Trabajo de Fin de Grado consiste en la obtención e implantación de un modelo de Inteligencia Artificial para la emulación razonable de los pintores impresionistas Monet, Van Gogh y Cézanne, de ser posible con el estado del arte actual. La implementación de dicho modelo suministrará una imagen de entrada, es decir, no se busca que suministre un cuadro aleatorio, sino un cuadro *inspirado* en una imagen ya existente, emulando a la pintura al aire libre característica del Impresionismo.

1.2.1. Objetivos secundarios

Además del objetivo principal, se han especificado los siguientes objetivos secundarios:

1. Realizar una reflexión sobre la capacidad actual de la Inteligencia Artificial respecto a la emulación de la creatividad propia de las Bellas Artes.
2. Aprender el lenguaje de programación Python y las plataformas Keras y Tensorflow.
3. Construir una arquitectura software que permita la incorporación de otros pintores de forma sencilla para el lector.
4. Explorar las posibilidades de la nube para el desarrollo de modelos de Inteligencia Artificial.
5. Elaborar una API REST para la implementación de servicios web para expandir las posibilidades del modelo construido previamente.
6. Aprender L^AT_EX para la realización del presente documento.

1.3. Estructura del documento

Tras esta introducción, el resto de este documento se estructura en los siguientes apartados:

- **Estado del arte:** Se expone el Impresionismo pictórico como corriente artística, junto a las características de los tres pintores que trataremos de emular mediante el modelo de Inteligencia Artificial. Por otra parte, se exponen los conceptos de Inteligencia Artificial, Machine Learning y Redes Neuronales, los diversos tipos de redes neuronales relacionados con la Visión Artificial y por último las Redes Generativas Adversativas Cíclicas, que es la solución propuesta para realizar este Trabajo de FIn de Grado
- **Tecnologías utilizadas:** En esta sección se exponen las tecnologías utilizadas para el desarrollo del proyecto de forma significativa, con la finalidad de que el lector comprenda qué son y por qué se han usado tales herramientas en la elaboración del proyecto.
- **Desarrollo del proyecto:** Este capítulo conforma el núcleo del presente documento: se aborda el diseño del sistema, los datos utilizados para entrenar el modelo, la documentación del sistema para que el lector pueda replicar el proyecto y realizar modelos propios y los resultados obtenidos.
- **Conclusiones:** Se desarrollan las conclusiones obtenidas de la investigación y del desarrollo del proyecto, además de la experiencia personal del autor en la realización del mismo.
- **Impacto social:** Este apartado contiene una reflexión sobre el impacto social y medioambiental de este Trabajo Fin de Grado.
- **Líneas futuras:** En esta sección se detallarán las posibles ampliaciones y mejoras del proyecto a juicio del autor.
- **Anexos:** En los Anexos se detallan los entornos de desarrollo utilizados en la elaboración del proyecto, así como el código fuente del mismo.

Capítulo 2

Estado del arte

En esta sección se exponen los fundamentos del Impresionismo pictórico y de la Inteligencia Artificial, para posteriormente centrarnos en las Redes Neuronales Artificiales y sus modernos sistemas con el objetivo de que el lector comprenda las técnicas utilizadas en este Trabajo Fin de Grado . Para ampliar conocimientos, remitimos a la bibliografía que citaremos a lo largo de este Capítulo.

2.1. Impresionismo Pictórico

Como ya comentamos brevemente en la introducción, el Impresionismo pictórico es un estilo originado en Francia en la segunda mitad del siglo XIX, que se caracteriza por su persistente experimentación con la iluminación [10], ya que se considera como un factor crucial para generar belleza.

2.1.1. Origen del nombre

En abril de 1874 se organiza la primera exhibición de un grupo de jóvenes pintores franceses, cansados del enfoque conversador que imperaba en la Académie des Beaux-Arts (Academia de Bellas Artes). Tras la exposición aparecen las primeras reseñas, siendo esta la que daría nombre al movimiento:

Impresión, desde luego eso produce. Simplemente me estaba diciendo que, ya que estaba impresionado, tenía que haber alguna impresión en ello . . . ¡y qué libertad, qué facilidad de fabricación! El papel tapiz en su estado embrionario es más acabado que ese paisaje marino.

Louis Leroy, sobre “Impresión, Sol naciente” (figura 2.1) [11]



Figura 2.1: Impresión, Amanecer, de Claude Monet (Musée Marmontel Monet, Paris). Extraído de [12].

Dicha crítica despectiva, en lugar de ridiculizar al movimiento, lo da a conocer y lo expande rápidamente.

2.1.2. Contexto artístico e histórico

Antes de la aparición del Impresionismo, el estilo predominante en Europa era el Realismo: abandonaron la nostalgia propia del Romanticismo en favor de proyectar los ideales en el porvenir. El hombre pasa a soñar partiendo de la realidad, por lo que el arte se torna *realista* [3].



Figura 2.2: Las espigadoras, de Jean-François Millet (Musée d'Orsay). Año 1857. Extraído de [13].

Entre finales del siglo XVIII y la primera mitad del siglo XIX el mundo sufre una enorme cantidad de transformaciones: la segunda revolución industrial, las revolución francesa, la Europa de la Restauración Absolutista y las reformas burguesas. La sociedad entra en ebullición a todos los niveles, apareciendo corrientes filosóficas como el Positivismo y el Socialismo, mientras se publican obras tan influyentes como el Manifiesto Comunista de Karl Marx (1848).

En mitad de esta vorágine, los impresionistas reaccionan dando un mayor peso a la luz y el color; aspectos que toman de la pintura veneciana de mediados del siglo XVI y de la pintura holandesa del siglo XVII.



Figura 2.3: Grupo familiar ante un paisaje, de Frans Hals (Museo Nacional Thyssen-Bornemisza). Pintura holandesa del año 1648. Extraído de [14].

2.1.3. Características artísticas

El Impresionismo se revela como una contestación al Realismo, ya que donde el Realismo presenta determinación, el Impresionismo muestra "lo continuo e indefinido, por la inestabilidad y el cambio perpetuo, incapaz de ofrecer una visión precisa de la realidad. De ahí las formas imprecisas, el toque distendido, la incertidumbre total" [3]. Ese cambio de filosofía se expresa con los siguientes rasgos [11]:

- La importancia de la luz: como ya hemos mencionado anteriormente, muchos autores practicaban la pintura al aire libre, por lo que pudieron estudiar de cerca la luz y sus efectos en la naturaleza.
- Pinceladas gruesas y pictóricas: los impresionistas emplean pinceladas gruesas, como si se tratase de un boceto. Esto permite capturar la naturaleza efímera y fugaz a la vez que experimentaban con el color y las formas que interactúan sobre el lienzo. Observe la figura 2.4.

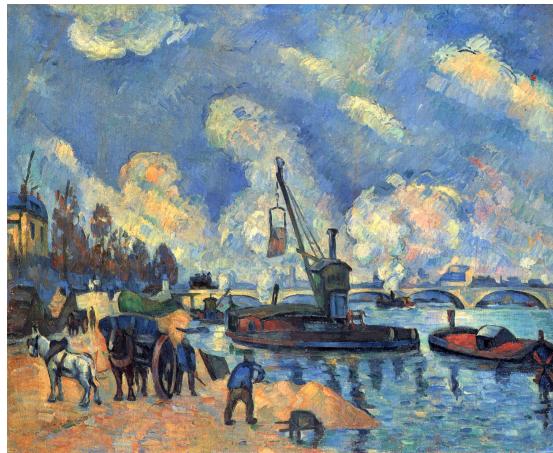


Figura 2.4: El Sena en Bercy, de Paul Cézanne. Extraído de [15].

- Una distintiva paleta de colores: en lugar de mezclar la pintura para obtener nuevos tonos, agrupaban pinceladas individuales de varios colores. Este método se puede observar en las representaciones de sombras y nieves, que nunca son completamente negras o blancas, respectivamente. También suelen presentar esquemas de colores neutros con toques vivos de rojo que atraen la vista y dan equilibrio a las composiciones. Puede apreciarse esta nueva paleta en la figura 2.5.
- Sujetos y temas cotidianos: influenciados por el costumbrismo español abanderado por Francisco de Goya, el contenido típico de las pinturas impresionistas incluye bodegones, paisajes, retratos, y escenas urbanas; nada que ver con las escenas históricas, mitológicas y alegóricas de la pintura francesa tradicional.



Figura 2.5: La terraza de Sainte-Adresse, de Claude Monet (Metropolitan Museum of Art, Nueva York). Año 1867. Extraído de [16].

2.1.4. Monet

Claude Monet (1840-1926), pintor francés, fue la figura clave del movimiento impresionista. Dibujante de caricaturas desde niño, ya pintaba paisajes y marinas, algo que le agradaba al poder trabajar al aire libre [17]. Su exilio a Argelia para evitar el servicio militar (si bien su familia pagó el reemplazo un año después y pudo regresar a Francia) y sus viajes por Europa hacen que se enamore de las distintas luces intríscicas a los momentos del día.



Figura 2.6: Retrato de Claude Monet, por Pierre-Auguste Renoir (Musée d'Orsay, París). Extraído de [18].

Sus innovaciones en el estudio de la luz y del color causaron reacciones mixtas, si bien se adelantó lo justo a su tiempo para ser considerado innovador

y tener éxito en vida (a diferencia de muchos otros pintores) al mismo tiempo.



Figura 2.7: Mujer con sombrilla mirando a la izquierda, de Claude Monet (Musée d'Orsay, Paris). Extraído de [19].

Si bien en sus primeros cuadros tenía un estilo excesivamente experimental y poco comercial, tras la exposición de 1874 en la *Société anonyme des artistes peintres, sculpteurs et graveurs* (Sociedad Anónima de pintores, escultores y grabadores), que cofundó junto a otros pintores, descubrió su estilo: captaba el instante y la luz a través de manchas, lo que junto a su radicalización con el tiempo provocó que en sus últimas obras la forma está ya prácticamente disuelta en manchas de color [17].

Máximo exponente de la pintura en *plein air*, o al aire libre [11], su estilo basado en el cambio del paisaje lo resumió en la siguiente frase.

Para mí, un paisaje no existe por derecho propio, ya que su apariencia cambia en cada momento; pero la atmósfera circundante le da vida –la luz y el aire que varían continuamente. Para mí, es solo la atmósfera circundante la que da a los sujetos su verdadero valor.

Claude Monet [11]

2.1.5. Van Gogh

Vicent Van Gogh (1853-1890), pintor holandés, comenzó su carrera profesional de pintor a los 32 años (de los 37 que vivió), pintando más de 900 cuadros y 1600 dibujos en sólo 5 años [20]. Conocido por su pintura atemporal y por sus problemas psiquiátricos (su famosa oreja cortada durante un enfrentamiento en 1888) a partes iguales, fue un audaz experimentador con sus pinceladas bruscas y sus colores chillones. A diferencia de Monet, Van Gogh

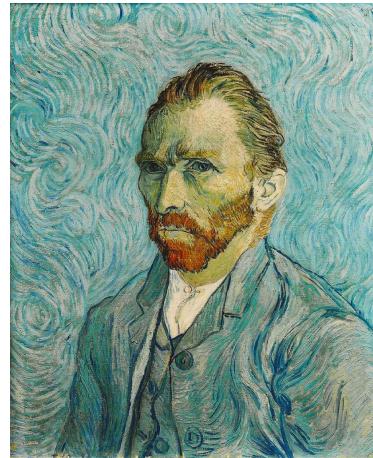


Figura 2.8: Autorretrato, de Van Gogh (Musée d'Orsay, Paris). Extraído de [21].

no vendió ningún cuadro en su vida, si bien se debía más a su personalidad que al reconocimiento de su arte.

Disfrutó de amistad y admiración con los mejores pintores de su época, reinventando el impresionismo previo de Monet (tanto él como Cézanne son considerados postimpresionistas, si bien por simplificación los incluimos como impresionistas). Esta evolución se puede ver en los colores vivos, en el abandono del naturalismo en sus composiciones, formas que parecen moverse o caerse (influyendo al posterior surrealismo).

Sus problemas psiquiátricos (con el ingreso de un año en un frenopático después de su incidente con la oreja) no le impidieron componer de forma asombrosa, con cuadros como la serie de *La noche estrellada*. Uno de ellos se muestra en la figura 2.9. Se suicidó en 1889 disparándose en el pecho con un revólver [22].

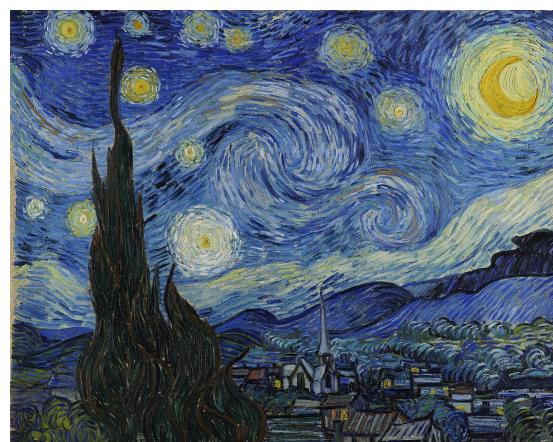


Figura 2.9: La noche estrellada, de Van Gogh (Museum of Modern Art, Nueva York) Extraído de [23].

2.1.6. Cézanne

Paul Cézanne (1839-1906), pintor francés. Pintor ignorado en vida, apenas expuso y pasó una vida enclavado en la pobreza, con poco reconocimiento en vida [24].

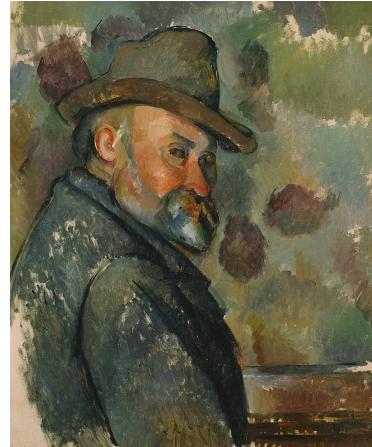


Figura 2.10: Autorretrato con sombrero arrugado, de Paul Cézanne (Bridgestone Museum of Art, Tokio). Extraído de [25].

Si bien frecuentaba los círculos impresionistas (expuso en 1874 en la exposición inicial del grupo), las críticas que recibieron sus cuadros provocaron que Cézanne no volviese a exponer, comenzando un camino personal. No obstante, hacia el final de su vida volvió a exponer, comenzando a ser su obra valorada e influyendo en los pintores posteriores [26].

Con el paso del tiempo, sus cuadros empezaron a ser muy valorados. Su cuadro más famoso, *Los jugadores de cartas* (figura 2.11) del año 1893, fue adquirido por la familia real de Catar por 250 millones de dólares (191,6 millones de euros) en 2011 [27].



Figura 2.11: Los jugadores de cartas, de Paul Cézanne (Colección privada de la familia real catari). Extraído de [28].

A diferencia de Monet y Van Gogh, Cézanne intentó aunar en sus obras el orden, la expresión personal y el naturalismo, estructurando la obra en formas simples y planos de color. Esto lo conseguía representando de la forma más exacta posible desde el punto de vista pictórico, con pinceladas muy particulares con las que representa simultáneamente lo que el ojo que observa y una abstracción de eso, es decir, perspectivas diferentes al mismo momento. Esta simultaneidad sería adoptada posteriormente por los pintores cubistas [24].

¡Cézanne es mi único y singular maestro!

Pablo Picasso en 1964 [29]

2.2. Inteligencia Artificial y Redes Neuronales

2.2.1. ¿Qué es la Inteligencia Artificial?

La Inteligencia Artificial es un concepto con numerosas definiciones e interpretaciones, por lo que no se pretende en este documento hacer un estudio extenso y profundo de la misma ya que se escapa a los límites y objetivos de este Trabajo Final de Grado. Para ampliar sobre este tema se recomienda acudir a la bibliografía citada en esta sección.

A la hora de definir la inteligencia artificial cabe esperar que primero haya que clarificar qué entendemos por inteligencia. Una de las muchas definiciones que tiene inteligencia es la siguiente:

Habilidad para responder flexiblemente a diferentes situaciones, saber aprovechar circunstancias fortuitas, dar sentido a mensajes ambiguos o contradictorios, encontrar similitudes entre situaciones diferentes y generación de nuevos conceptos e ideas innovadoras.

Douglas Hofstadter [30]

Basándonos en esa definición, podemos describir la inteligencia artificial de la siguiente manera:

El arte de crear máquinas que realicen funciones que requerirían inteligencia si fueran realizadas por humanos.

Raymond Kurzweil [30]

Una vez clarificado qué entendemos por Inteligencia Artificial, podemos ver que el concepto es bastante amplio, por lo que se puede subdividir el concepto en diferentes niveles para ayudar a situarnos [31]:

- Inteligencia artificial débil: este tipo de inteligencia es capaz de resolver problemas muy bien definidos y acotados. En esta división es donde se enmarcan los progresos hasta la fecha en el campo, mediante técnicas como el *machine learning* (aprendizaje máquina) y *deep learning* (aprendizaje profundo) para resolver problemas específicos. Se ha conseguido que estos

sistemas acaben realizando esas tareas mucho más rápido (y en ocasiones mejor) que un ser humano, pero no son capaces de adaptarse a su entorno. Este Trabajo de Fin de Grado se ubica en esta sección.

- Inteligencia artificial general: esta inteligencia artificial, aún inexistente, permitiría resolver cualquier tarea intelectual resoluble por un ser humano, es decir, sería multitarea como los humanos. No sería únicamente una concatenación de inteligencias artificiales débiles, ya que además sería capaz de realizar juicios, razonar ante la incertidumbre, planificar y aprender.
- Inteligencia artificial fuerte: añadiría a la general la conciencia de sí misma, muy en la línea de *Skynet* y las demás inteligencias artificiales planteadas por el cine. Teóricamente sería capaz de resolver cualquier problema y sentiría emociones.

2.2.2. Fundamentos de Machine Learning

El *Machine Learning* (o Aprendizaje Automático en castellano) es un paradigma en el que el programador no indica las reglas que debe ejecutar el programa, sino que suministra los datos a tratar y (en ocasiones) las respuestas deseadas; y el sistema debe aprender las reglas que resuelven el problema.

Este proceso se repite durante un número concreto de veces (llamado *epoch*), para posteriormente evaluar el modelo en otro conjunto de datos (independiente del primero), llamado conjunto de test, de acuerdo a unas métricas.

En ocasiones se posee un tercer conjunto (independiente de los dos anteriores), llamado conjunto de validación, sobre el que se realiza una evaluación con el modelo *más refinado posible* para comprobar si es válido en la vida real y si es así ponerlo a funcionar.

Por otra parte, el proceso mediante el cual el sistema aprende se puede dividir en cuatro tipos:

- Aprendizaje supervisado: el programador proporciona al sistema un conjunto de datos, para los que se conoce tanto la entrada como la salida. Un ejemplo de este tipo son los filtros de spam.
- Aprendizaje no supervisado: en este caso el sistema conoce las entradas pero no las salidas del mismo. Se suele utilizar para comprender, entender o eliminar datos (*clustering*) y para visualización y reducción de dimensionalidad (Análisis por Componentes Principales).

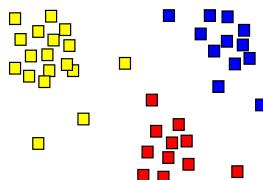


Figura 2.12: Ejemplo de clustering. Imagen de dominio público.

- Aprendizaje semisupervisado: en lugar de disponer para cada dato su salida esperada (o no disponer de ella), se tienen los dos casos. Si bien suena extraño, hay ejemplos en nuestra vida cotidiana: servicios como Google Fotos pueden detectar las personas presentes en una galería de fotos, pero el usuario debe decirle al sistema quién es [32]. La mayoría de estos algoritmos suelen ser mezcla de modelos supervisados y no supervisados, como en el caso de las *Deep Belief Networks*.
- Aprendizaje por refuerzo: en este caso, el sistema (o agente en la literatura de este campo) observa el entorno y realiza una serie de decisiones, por las que recibe una recompensa (puede ser negativa). En este caso, el agente tiene que aprender cuál es la mejor estrategia para maximizar la recompensa obtenida. Este tipo de aprendizaje se aplicó en la máquina AlphaGo (de la que hablamos en la introducción), o en algo mucho más común como el adiestramiento de una mascota.

2.2.3. ¿Por qué usar Machine Learning?

Para dar convencer al lector, usaré el ejemplo mostrado en [32]. Supongamos que queremos realizar un sistema de detector de spam mediante programación tradicional, el cual no dispone de una solución cerrada mediante un algoritmo. Seguiríamos el siguiente proceso para realizar el programa:

1. Analizaríamos los rasgos que presentan los mensajes de spam. Podríamos empezar con palabras tales como tarjeta de crédito; gratis; alucinante; para posteriormente complicar el sistema incluyendo reglas sobre el nombre del emisor, el cuerpo del mensaje...
2. Implementaríaos dichos rasgos en un programa que tomase un correo y ejerciese de filtro, indicando si el filtro es (o no) spam.
3. Testearíamos el programa y repetiríamos los pasos 1 y 2 hasta que el programa sea lo *suficientemente bueno*.

Debido a la complejidad del problema, el conjunto de reglas del problema sería cada vez mayor y más complicado, siendo difícil de mantener. En contrapartida, un sistema basado en aprendizaje automático captaría los patrones de palabras y párrafos propios de los mensajes de spam, por lo que el programa sería mucho más fácil de desarrollar y mantener.

Una vez lanzásemos nuestro programa, las personas que envían spam se darían cuenta de qué palabras bloquean sus mensajes, por lo que modificarían los mismos para saltarse nuestro filtro, por lo que tendríamos que reescribir las reglas indefinidamente. Sin embargo, el sistema basado en *Machine Learning* aprendería las nuevas técnicas de los *spammers* sin nuestra intervención, ahorrando desarrollo y problemas al programador.

Este enfoque se puede extender a todos los problemas que no disponen de una solución algorítmica sencilla: donde un programa tradicional tendría que desarrollar un algoritmo con reglas muy concretas y focalizadas para cada casuística, un sistema de aprendizaje automático puede reconocer esos patrones

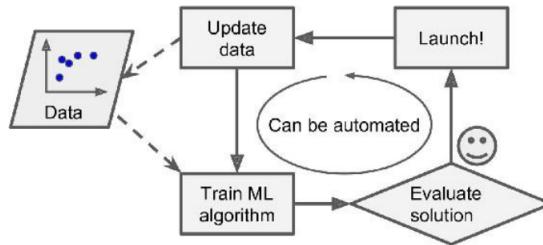


Figura 2.13: Ciclo de vida de una aplicación de Machine Learning. Tomado de [32].

para todo el conjunto de datos del que dispongamos. Este descubrimiento de patrones nos permite detectar correlaciones poco imaginables a priori, además de nuevas tendencias, lo que nos permite entender mejor nuestro problema. Este último concepto se conoce como Minería de Datos.

No obstante, dentro del Machine Learning también existen problemas. Entre ellos destacamos :

- Cantidad insuficiente de datos: a diferencia de los humanos, las máquinas no aprenden únicamente con un sólo ejemplo, necesitan ver miles o millones de ejemplos durante cientos de veces para aprender. Disponer de pocos datos produce ruido muestral, entre otros fenómenos.
- Sesgos en la elaboración del conjunto de datos y/o incompletitud del mismo. Como en nuestra vida, si disponemos de datos sesgados y/o incompletos no podemos realizar razonamientos correctos.
- Overfitting o sobreajuste: existe el riesgo de que el modelo pierda la capacidad de generalizar y que simplemente se aprenda el conjunto de entrenamiento. Por este fenómeno se elige el modelo mediante sus resultados en el conjunto de test. Puede mitigarse reduciendo el número de parámetros del modelo y/o aumentando la cantidad de datos y/o la calidad de los mismos.

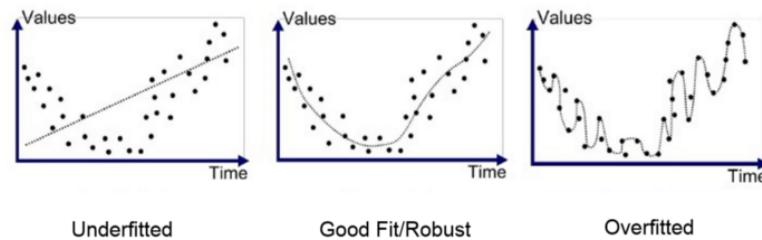


Figura 2.14: Ejemplo de underfitting, ajuste óptimo y overfitting. Extraída de [33]

- Underfitting o infrajuste: el sistema es incapaz de aprender y generalizar. Ocurre al seleccionar un modelo demasiado simple, aunque también puede

ocurrir si las características que utiliza el modelo para aprender son poco potentes.

2.2.4. ¿Qué son las Redes Neuronales?

Una red neuronal es un sistema computacional inspirado en las neuronas biológicas que cada uno disponemos en nuestro cerebro, creado por los investigadores Warren McCulloch y Walter Pitts en 1943 [34].

Una neurona artificial se compone de neuronas y conexiones, siendo la neurona ^aislada un equivalente artificial a las dendritas y axones; mientras que el enlace simula la sinapsis biológica al interconectar varias neuronas en un enlace de 1 a varios (la salida de una neurona se interconecta a cierto número de neuronas) [35]. Matemáticamente la neurona realiza la siguiente operación:

$$y = \sigma\left(\sum_{i=1}^n (x_i \cdot \omega_i) + b\right) \quad (2.1)$$

donde y es la salida de la neurona, σ es la función de activación de la neurona, x_i es la entrada i -ésima de la neurona, ω_i es el peso (ponderación) del enlace de la neurona i -ésima y b representa al término *bias*, un parámetro por el cual el modelo puede desplazar o ajustar una función de activación sin transformar su forma por completo.

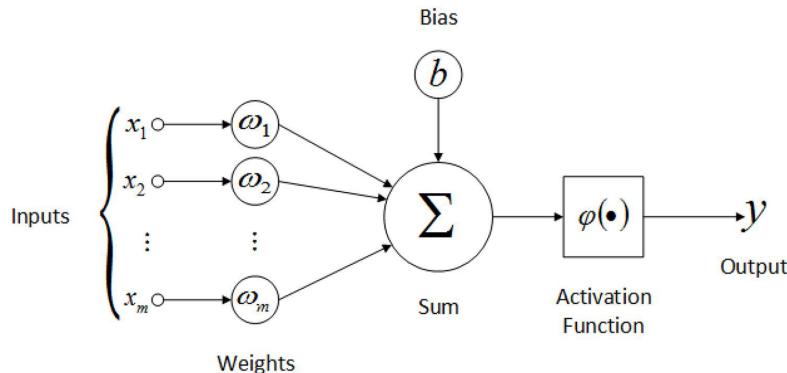


Figura 2.15: Representación gráfica de la función de una neurona artificial realizando la ecuación 2.1. Extraída de [36].

Ampliemos ahora el significado de algunos elementos de la ecuación 2.1:

2.2.4.1. Pesos de las neuronas y algoritmo básico de aprendizaje

Los pesos de las neuronas (ω_i) son números reales, utilizados por el modelo para su ajuste, inicializándose aleatoriamente. El método por el cual la neurona aprende los valores correctos de los pesos es minimizando la función pérdida, que mide la diferencia entre la salida esperada y la salida actual. Dicha diferencia puede calcularse mediante muchas funciones, como la función *Root Mean Square Error* (RMSE).

$$RMSE = \sqrt{\frac{1}{N} \cdot \sum_{n=1}^N (d_n - s_n)^2} \quad (2.2)$$

El ajuste de los pesos como tal es realizado por un algoritmo llamado optimizador, uno de los parámetros del modelo que determina el programador. Los más conocidos son el algoritmo clásico de *backpropagation* (retropropagación en inglés), que se basa en el método *descenso del gradiente* y ADAM , basado en *backpropagation* y en otros muchos. En este Trabajo Fin de Grado se utiliza ADAM debido a sus ventajas sobre otros optimizadores [37].

Por otra parte, los pesos son actualizados en cada iteración proporcionalmente a un parámetro llamado μ (ajustado por el programador) conocido como ratio de aprendizaje. Una vez modificados los pesos se realiza una nueva iteración con otros datos del conjunto de entrenamiento. Una vez se pasa por todos los datos, se vuelve a iterar sobre el mismo, así hasta alcanzar un número fijado de veces llamado *epoch*.

2.2.4.2. Funciones de activación

La función de activación (que puede ser cualquier tipo de función) es la encargada de introducir relaciones no lineales entre las variables de entrada y la de salida. Es un parámetro que también es fijado por el programador, ya que dependiendo de la tarea a realizar algunas tienen mejor desempeño que otras. Presentamos algunas de las más conocidas [38]:

Función	Expresión	Rango
Lineal	$f(x) = x$	$(-\infty, \infty)$
Escalón	$f(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ 1 & \text{si } x \geq 0 \end{cases}$	$(0, 1)$
Sigmoidal	$f(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
ReLU (Rectified Linear Unit)	$f(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ x & \text{si } x \geq 0 \end{cases}$	$[0, \infty)$
Leaky-ReLU	$f(x) = \begin{cases} 0,01 & \text{si } x \leq 0 \\ 1 & \text{si } x \geq 0 \end{cases}$	$(-\infty, \infty)$

Tabla 2.1: Comparación entre algunas de las funciones de activación más populares

2.2.4.3. Concepto de capa y de Red Neuronal Profunda

Hasta ahora sólo habíamos hablado de una neurona, sin tener en cuenta la arquitectura de la red a la cual pertenece. Existen dos categorías principales de estructuras de redes neuronales: acíclicas o redes con alimentación-hacia-delante y cíclicas o redes recurrentes [39]. Las primeras son las que explicamos en 2.1, y

las redes recurrentes se caracterizan porque las entradas dependen de sus salidas en un estado anterior, lo que las hace disponer de memoria a corto plazo pero son mucho más complicadas de entender, programar y entrenar, por lo que están fuera del alcance de este Trabajo Fin de Grado .

Las redes neuronales con alimentación hacia delante normalmente se organizan en capas, de forma que cada unidad recibe entradas únicamente de las unidades de la capa que la precede inmediatamente [39]. A las redes neuronales que sólo tienen una capa de entrada y otra de salida se les conoce como perceptrones.

El problema de los perceptrones es que sólo pueden aprender funciones lineales, y para que una red neuronal pueda aprender funciones no lineales (algunas tan simples como la XOR) es necesario incluir una o más capas intermedias, llamadas capas ocultas [40]. A las redes con una o más capas ocultas se les conoce como Redes Neuronales Profundas.

En la figura 2.16 se puede ver visualmente la diferencia, siendo la capa morada de la capa oculta.

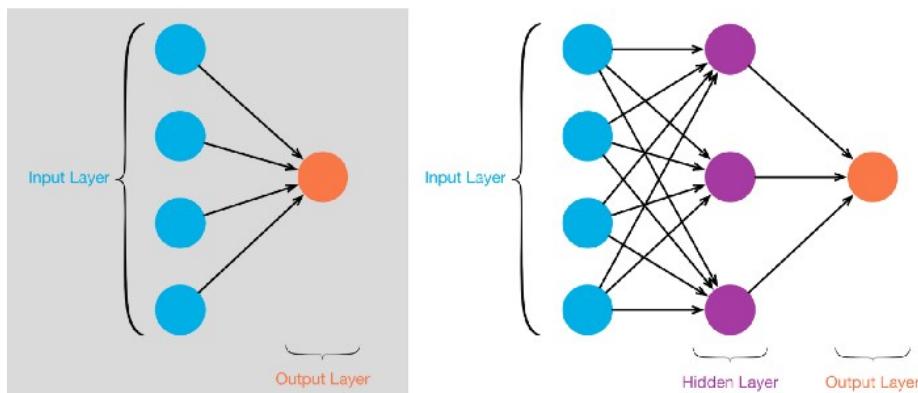


Figura 2.16: Perceptrón y Red Neuronal Profunda. Tomado de [41].

2.2.5. ¿Por qué usar Redes Neuronales?

Las Redes Neuronales son adecuadas para aquellos problemas que no tienen una solución computacional precisa o que requieren algoritmos muy extensos [34]. El problema que queremos resolver en este Trabajo Fin de Grado no tiene solución mediante un algoritmo sencillo y preciso, por lo que de primeras parecen adecuadas. Asimismo, las Redes Neuronales presentan las siguientes ventajas [35]:

- Están basadas en la inducción y la generalización. Aprenden de la experiencia.
- Se adaptan muy bien a la computación en paralelo, permitiendo una gran escalabilidad en problemas complejos.

- Autoorganización: crean su propia representación de la información que reciben durante el entrenamiento.
- Tolerancia a fallos: pueden trabajar con información parcial debido a que codifican la información de modo distribuido y redundante.
- Flexibilidad: pueden adaptarse a un nuevo entorno sin necesidad de ser reprogramadas, gracias a su capacidad de aprendizaje

No obstante, las redes neuronales también presentan desventajas:

- Elecciones de arquitectura y parámetros por ensayo y error si no hay estudios previos sobre problemas similares.
- Aprendizaje lento y dificultoso.
- Ausencia de explicación de las decisiones, lo que impide que comprendamos correctamente qué hace el sistema.

Podemos concluir que al elegir una red neuronal, no obtendremos muchos datos sobre cómo aprende el sistema y cómo representa la información internamente, pero tendremos una modelo que si lo conseguimos entrenar correctamente será eficiente y podrá aprender los estilos de Cézanne, Monet y Van Gogh programándolas una sola vez.

2.2.6. Redes Neuronales Convolucionales

Las Redes Neuronales Convoluciones son un tipo de red neuronal inspirada en la corteza visual que hay en los cerebros biológicos. Simplificando mucho, en la capa primaria o V1 de la corteza visual, tenemos células que reconocen patrones elementales (líneas horizontales y verticales) y que sólo se activan si lo reconocen [42]. Matemáticamente ese tipo de reconocimiento forma parte de una operación más genérica, llamada convolución (de ahí su nombre).

Las siguientes capas (V2, V3...) están conectadas a la salida de la capa inmediatamente anterior, reconociendo patrones cada vez más complejos [32]. Esto inspiró a Kunihiko Fukushima, quien presentó en 1980 el Neocognitron [43], que gradualmente evolucionó hasta las redes neuronales convolucionales de hoy día.

La idea principal de estas redes es que cada neurona de la capa de entrada aprende a reconocer un tipo de patrón sencillo, para que la siguiente capa, a partir de los resultados de la capa precedente, aprenda un patrón un poco más complejo que el anterior. Esta estructura jerárquica es muy común en las imágenes reales, por lo que las RNC funcionan muy bien en el reconocimiento de imágenes [32]. Finalmente, podemos definir que una red convolucional es una red profunda que consta de capas convolucionales y de reducción alternadas, con una capa de salida que realiza una convolución especial que *aplana* la matriz anterior (pasamos de una matriz multidimensional a una unidimensional).

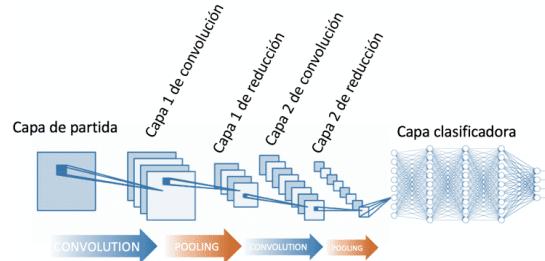


Figura 2.17: Arquitectura de una RNC. Tomada de [44].

2.2.6.1. ¿Qué es la convolución en reconocimiento de imágenes?

La operación de convolución extrae partes del mapa de entrada (el primer mapa es la propia imagen) y realiza una transformación a todos las secciones, produciendo un mapa de salida. Esta transformación consiste en multiplicar el mapa de entrada (que no deja de ser una matriz) por otras pequeñas matrices, denominadas kernel (filtro).

Este kernel se aplica a todas las neuronas de entrada (*deslizando una ventana*, marcada en rojo en la figura 2.18), generando una matriz de salida llamado mapa de características. Posteriormente a este mapa de salida se le aplica una función de activación, como la ReLU o la Leaky ReLU que expusimos en la tabla 2.1.

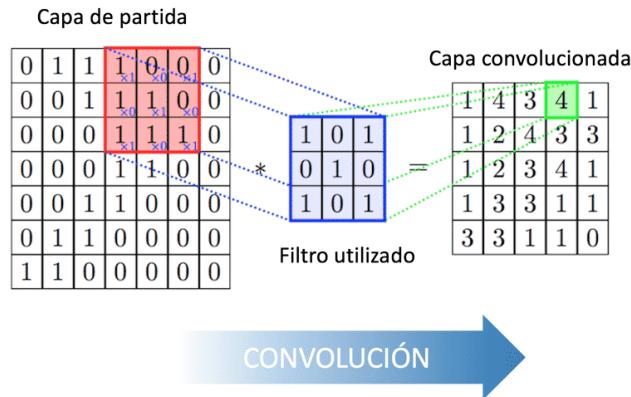


Figura 2.18: Operación de convolución]. Tomado de [44].

Cabe mencionar que los kernel no son fijos, sino que son aprendidos por la red neuronal, es decir, los valores de los kernel son los valores ω_i de la ecuación 2.1

2.2.6.2. ¿Qué es la reducción en reconocimiento de imágenes?

La reducción es una operación muy sencilla: para una matriz de entrada queremos reducir su dimensionalidad. Para ello dividimos la matriz en pequeñas

submatrices a las que les realizamos una operación, como pueda ser la media o el valor máximo.

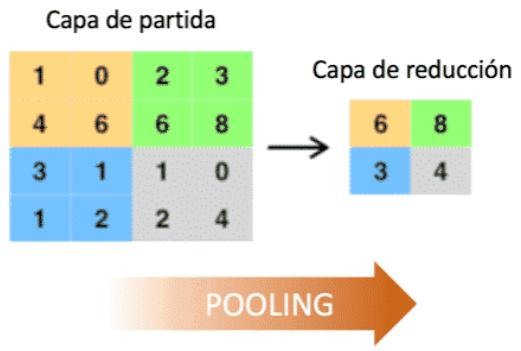


Figura 2.19: Operación de reducción. Tomado de [44].

Si usamos la operación valor máximo, prevalecen las características más importantes del filtro, reduciendo el tamaño del mapa a la mitad o más, reduciendo el número de pesos a aprender. Dicha operación se conoce como *max pooling*, representada en la figura 2.19

2.2.7. Redes Generativas Antagónicas

Las Redes Generativas Antagónicas, también conocidas como *Generative Adversative Network* o por sus siglas en inglés GAN (así serán nombradas de ahora en adelante) es un tipo de red neuronal bastante reciente (2014) [45], pero al mismo tiempo muy conocida debido a su inmenso potencial. Mucho se ha hablado de ellas por su concepto transgresor y revolucionario, ya que permite dotar a las redes neuronales de cierta *creatividad*. La figura 2.20 muestra 15 rostros generados por una red GAN.



Figura 2.20: Fotos de rostros humanos creados por una red GAN. Tomado de [46].

Realmente implementan una idea muy sencilla: pongamos a competir dos redes neuronales entre sí para que aprendan una de la otra indefinidamente.

Una de ellas, llamada generador, generará una imagen falsa, mientras que la otra red, llamada discriminador, decide si la imagen es real o no. La figura 2.21 muestra dicho esquema.

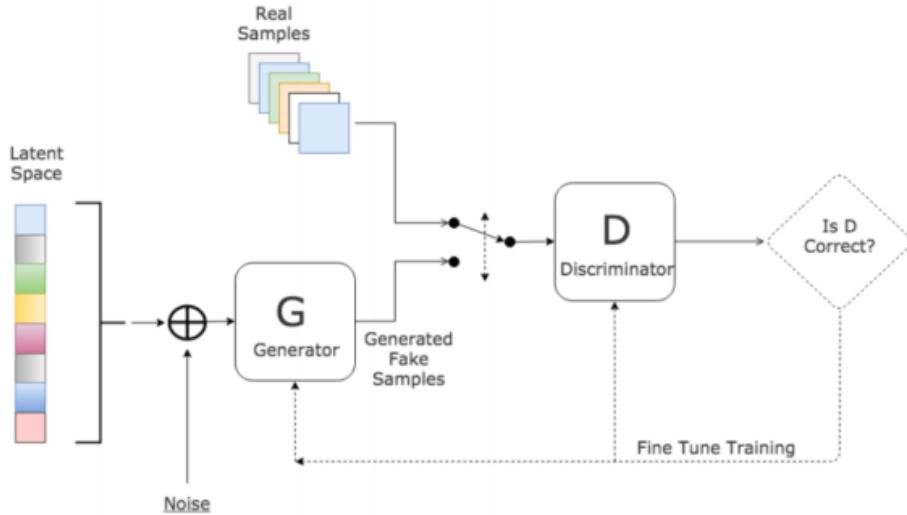


Figura 2.21: Esquema de una red GAN. Tomado de [47].

Esta competición tiene suma cero, es decir, el acierto de una de ellas se compensa con la pérdida de la opuesta, de forma que la que ha *perdido* aprende de ello. Eso nos permite refinar el modelo *indefinidamente*, ya que ambas mejoran continuamente. Por ejemplo el generador aprenderá a generar imágenes más precisas, mientras el discriminador aprenderá cada vez mejor los rasgos de las fotos para evitar ser engañada.

2.2.7.1. Generador: función y algoritmo de entrenamiento

El generador toma una distribución aleatoria como entrada (normalmente Gaussiana) y genera una imagen salida [32]. Esta imagen comenzará siendo ruido aleatorio, para empezar a ofrecer resultados cada vez más realistas.

Esta red se entrena después del discriminador y de forma totalmente diferente: el generador produce un conjunto exclusivo de imágenes falsas, las etiquetamos como imágenes reales y se las pasamos al discriminador (le intentamos engañar). El error cometido por el discriminador clasificando las imágenes falsas es el que usamos para entrenar el generador. Por ejemplo, si el discriminador ha etiquetado todas las imágenes como reales, entrenamos al generador con error 0 (es decir, no lo modificamos) puesto que le engaña perfectamente. En este paso los pesos del discriminador son congelados para que el entrenamiento afecte únicamente al generador.

2.2.7.2. Discriminador: función y algoritmo de entrenamiento

El generador únicamente se encarga de discernir si la imagen recibida es verdadera o falsa. Dicha imagen puede ser del conjunto de datos reales o del

generador.

El entrenamiento de esta red se hace antes que el del discriminador, y se realiza como sigue: en primer lugar se compone un conjunto de imágenes para que las vea el generador: la mitad será del conjunto de imágenes real y la otra mitad del generador y a diferencia del caso anterior los etiquetamos como corresponde. Evaluamos los resultados, calculamos el error y el optimizador se encarga de modificar los pesos para la siguiente iteración. Es decir, su algoritmo de entrenamiento es el tradicional. Cabe decir que como en el caso del discriminador, los pesos de la otra red se congelan para que se entrene únicamente el generador.

2.2.7.3. Las dificultades del entrenamiento de las redes GAN

Aunque a priori el juego de la competición entre redes puede dar muy buenos resultados, no los produce en todos los casos. Existe un estado final, llamado *equilibrio de Nash* en el que ningún jugador mejora su comportamiento si cambia su estrategia si se asume que el rival no cambiará la suya. Por ejemplo, el equilibrio de Nash aparece en la vida real: los conductores circulan por el lado derecho de la calzada y ningún conductor *mejora* por ir al lado izquierdo, pero también puede darse que los conductores aprendieran a conducir por el lado izquierdo y ninguno circulase de manera correcta [32].

Para evitar los efectos adversos de este fenómeno los autores de las redes GAN demostraron que las redes GAN sólo producen un único equilibrio de Nash, y lo consiguen haciendo que las imágenes que recibe el discriminador en el entrenamiento del generador sean 50% reales y 50% falsas. Por otra parte esto no garantiza que la red alcance este equilibrio, además de que incrementa el tiempo de entrenamiento.

No es la única dificultad que tienen: su mayor problema es el denominado *modo colapso* o *mode collapse* en inglés: se produce cuando el generador va perdiendo diversidad en sus salidas, perdiendo gradualmente toda su *creatividad*.

Para explicar este fenómeno utilizaré el ejemplo presentado en [32]: supongamos que tenemos un conjunto de datos que sean prendas de ropa y el generador produzca mucho mejor zapatos que otra cosa. Como engaña mejor al discriminador que produciendo otra cosa, el generador comenzará a generar cada vez más zapatos, olvidando cómo se generan otras prendas de ropa. Lo mismo le pasará al discriminador: se centrará tanto en describir zapatos que olvidará cómo son las otras prendas. Llegará un momento en el que el discriminador sepa perfectamente cómo distinguir los zapatos reales de los falsos, por lo que el generador buscará otra prenda de ropa (por ejemplo camisetas), olvidando gradualmente cómo eran los zapatos y así sucesivamente.

Esto provoca que a veces la red GAN no converja o que su entrenamiento sea muy inestable. Como esto se produce por numerosos factores, las redes GAN sean muy sensibles a los hiperparámetros (el conjunto de parámetros fijado por el programador).

Por tanto, vemos que las redes GAN presentan una serie de problemas, si bien los investigadores están proponiendo nuevas técnicas para solucionarlas. Como esas soluciones son dependientes del problema y no son necesarias para entender las redes GAN, no las incluimos en este documento.

2.2.8. Transferencia de estilo y Cycle GAN

2.2.8.1. Transferencia de estilo

La transferencia de estilo es el concepto que queremos implementar en este Trabajo Fin de Grado, ya que el concepto designa a un sistema que es capaz de transformar una imagen de entrada a otro dominio diferente [48]. Aquí toca realizar una matización: el objetivo de este Trabajo Fin de Grado es emular el estilo propio de un pintor, no de un sólo cuadro del pintor.

Esta técnica nos permite cambiar el dominio por otro cualquiera, por lo que el sistema es válido para otro pintor, únicamente cambia el entrenamiento del mismo. Esto permite a que el lector pueda implementar sus propios modelos a partir de sus propias imágenes u obteniendo los cuadros de otros pintores.

Las dos técnicas más populares dentro del campo de la transferencia de estilo son las Redes Generativas Antagónicas Cíclicas y la Transferencia de Estilo Neuronal.

La primera de ellas es la que implementaremos en este Trabajo Fin de Grado, ya que el conjunto de destino puede tener un número arbitrario de fotos y permite que el sistema aprenda el estilo del pintor; mientras que la Transferencia de Estilo Neuronal (*Neural Style Transfer* en inglés) únicamente permite el aprendizaje de una sola obra.

La figura 2.22 muestra un montaje realizado y cedido por Juan Luis Moreno Sancho, amigo del autor, realizado mediante *Neural Style Transfer*.

2.2.8.2. Cycle GAN

Las Redes Generativas Antagónicas Cíclicas, conocidas como *Cycle GAN*, es una red que consiste en dos redes GAN intercomunicadas entre sí [49]. Este sistema permite la implantación de nuestro objetivo principal: la conversión de fotos a cuadros, pero *Cycle GAN* es más que eso: permite convertir los dos dominios de forma bidireccional, lo que nos permite pasar de fotos a cuadros y de cuadros a fotos (es decir, el paisaje que trajeron de *imitar* los pintores).

Además de la bidireccionalidad de las conversiones, su mayor ventaja frente a otros modelos como *pix2pix* es que no se necesitan pares de ejemplos¹. Esta característica es la que hace posible que nuestro objetivo pueda implementarse, ya que no disponemos, por ejemplo, de los paisajes que veía Monet en sus

¹Por pares de ejemplos nos referimos a disponer del mismo elemento en ambos dominios. Un par podría ser las palabras *hola* en castellano y *hello* en inglés



Figura 2.22: Montaje de Zelda y La noche estrellada. Obra de Juan Luis Moreno Sancho.

cuadros.

Por último, una *Cycle GAN* son dos redes GAN unidas entre sí. Por tanto tenemos dos generadores y dos discriminadores. Uno de los generadores pasa del dominio del pintor al fotográfico, mientras que el otro hace la transformación inversa. Al no tener pares de ejemplos para entrenar nuestros generadores, tenemos que entrenar los dos discriminadores para ver si las imágenes son válidas o no, tal y como hacíamos en una red GAN cualquiera. Por tanto, tenemos un discriminador que será capaz de distinguir fotografías de falsificaciones, y otro que hará lo mismo pero con los cuadros. [48]

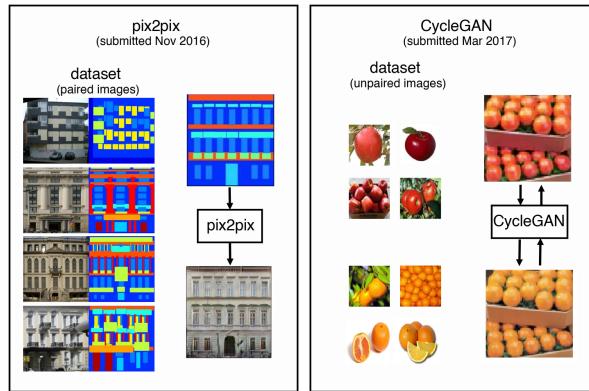


Figura 2.23: Comparativa entre pix2pix y Cycle GAN. Extraido de [48]

Capítulo 3

Desarrollo del proyecto

3.1. Características técnicas de los modelos a implementar

3.1.1. Cycle GAN

En primer lugar, una vez explicado el concepto de Cycle GAN en la sección 2.2.8.2, tenemos que mencionar las arquitecturas de las redes discriminadoras y generadoras que vamos a implementar. En el paper original [49] se proponen dos arquitecturas diferentes para el generador (U-Net y ResNet) y una para el discriminador (PatchGAN). Explicamos el fundamento de cada una a continuación.

3.1.1.1. U-Net

Como puede apreciar el lector en la figura 3.1, el nombre de red u no es casualidad.

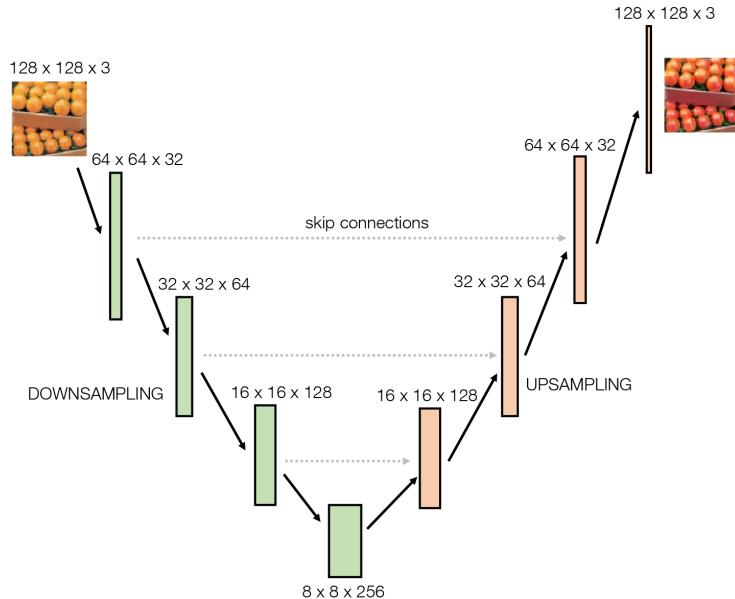


Figura 3.1: Arquitectura U-net. Extraido de [48]

En la primera mitad de la U se realizan operaciones en las que se reduce la imagen (*downsampling*) espacialmente pero se aumenta su número de canales y la segunda parte, la cual realiza la operación contraria (*upsampling*): aumenta la resolución espacial pero reduce el número de canales. No obstante, el flujo de la

Imagen es lineal: tenemos *atajos* o *skip connections* que comunican dos capas no contiguas, lo que permite disponer de más información en las capas de aumento.

Esta arquitectura sintetiza lo siguiente: las operaciones de reducción reconocen lo que hay en la imagen pero no aportan información sobre dónde está lo que reconocen. En el vértice de la U se conoce qué es lo que hay en la fotografía, pero no de su localización espacial; por lo que si simplemente quisieramos realizar segmentación de la imagen ya habríamos acabado.

Como no queremos únicamente segmentar la imagen, al ampliar la resolución (segunda parte de la U) tenemos que *indicar* a la capa dónde tiene que buscar lo que ya sabe. Esta información es aportada mediante los *atajos*, lo que nos permite combinar la información relacionada con el *qué hay* en la imagen junto con el *dónde está* [50] [48].

3.1.1.2. ResNet

Esta arquitectura es bastante parecida en concepto a la anterior, ya que también permite el paso de información entre capas no continuas a través de *atajos*; pero su implementación es diferente: en lugar de realizar una U para conseguir esos *atajos*, se aplana esa U (pasando a un flujo completamente lineal) y se realizan menos operaciones de aumento y reducción; pero colocando entre medias bloques residuales (de ahí el Res del nombre). La figura 3.2 muestra la arquitectura resnet.

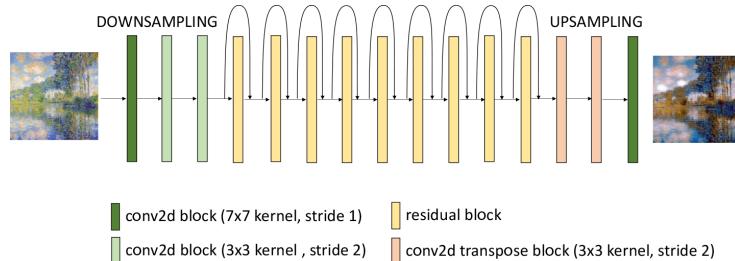


Figura 3.2: Arquitectura resnet. Extraido de [48]

Esos bloques residuales disponen de un *atajo*, permitiendo sumar al final del bloque los datos que ya teníamos en la entrada junto con el resultado de la capa actual. La figura 3.3 lo muestra gráficamente [51] [48].

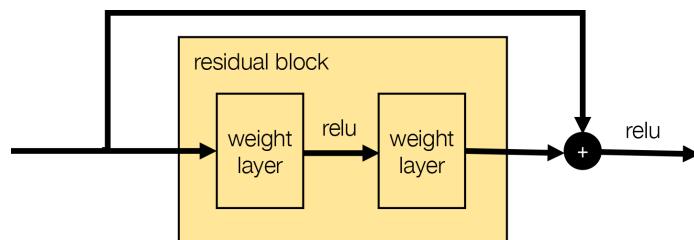


Figura 3.3: Bloque resnet. Extraido de [48]

La utilidad de esta arquitectura es que permite añadir cientos y miles de capas sin sufrir el problema de desvanecimiento de gradiente¹. Asimismo, se cree que añadiendo capas adicionales los resultados serán como mínimo igual de buenos, ya que si la capa añadida no es capaz de extraer información relevante, al sumar los resultados con los de la entrada del bloque residual seguimos conservando los datos pre-capas.

3.1.1.3. PatchGAN

Este discriminador, heredado de la red homónima, parte de una idea bastante sencilla de comprender [48]: en lugar de tener un discriminador que evalúa la imagen por completo y tiene por salida un número que indica la probabilidad de que la imagen de entrada sea *real*; PatchGAN propone un discriminador [52] (basado en la red convolucional VGG16 [53]) que proporciona una salida tensorial de 8x8 (calculado en paralelo) en lugar de un solo número. Ese 8x8 corresponde a que la imagen se divide en ese número de secciones, siendo cada escalar del tensor la probabilidad estimada de ese trozo de imagen.

Este enfoque permite que la función pérdida puede centrarse el estilo de la imagen (presente en cada sección de la imagen) en lugar del contenido (disperso a lo largo de las subimágenes), es decir, forzamos al discriminador a tener que fijarse en el estilo y no en los elementos de la imagen, que es lo que queremos discriminar en este problema.

3.1.2. Métricas de la CycleGAN

Como en cualquier modelo de redes neuronales la función pérdida es muy importante. Sin profundizar en las expresiones matemáticas, el modelo CycleGAN establece tres criterios diferentes:

1. Validez: ¿las imágenes creadas por cada generador engañan al discriminador correspondiente? Necesitamos que en este baremo el modelo lo haga realmente bien.
2. Reconstrucción: si pasamos por un generador y luego el otro (en ambos sentidos), es decir, realizamos un ciclo por la red, ¿tenemos la imagen original? Aquí necesitamos que el error sea mínimo.
3. Identidad: ¿si al generador le pasamos una imagen del mismo dominio (es decir, tenemos que el dominio origen y destino es el mismo), tenemos la imagen original? Nuevamente buscamos que este error sea lo más pequeño posible.

Las medidas de validez y reconstrucción suenan muy lógicas: una es el objetivo de la red y la otra asegura que los dos sentidos funcionen correctamente, pero la medida de identidad puede parecer poco importante. Para demostrar su importancia recurrimos al ejemplo mostrado en [48]: si prescindimos del baremo identidad, vemos que se modifica la superficie de las frutas. Esa es la labor de

¹En algunos casos, el gradiente de la función error tiende a valores muy pequeños, lo que en los peores casos impide que la red continúe aprendiendo debido a que se queda *atascada*, recibiendo únicamente actualizaciones de sus pesos con valor 0.



Figura 3.4: Medidas de validez y reconstrucción de la implementación con el dataset *vangogh2photo*, ofrecidas por Tensorboard

la métrica de identidad: forzar al sistema que sólo modifique las partes de la imagen que son necesarias para la transformación. Sin embargo, las tres partes de la función error tienen que estar balanceadas: si la parte identidad tiene poco peso tendremos el fenómeno visto en el ejemplo, pero si es demasiado el sistema está sentenciado a la inanición.

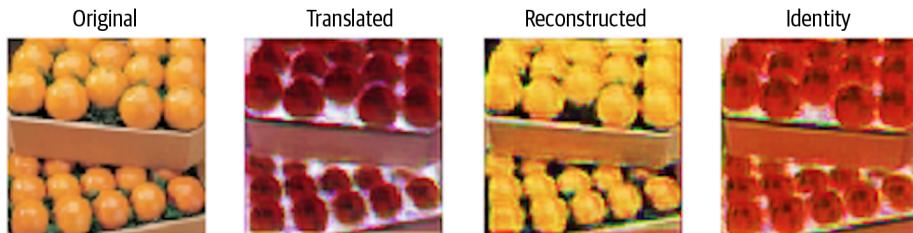


Figura 3.5: Importancia del error identidad. Tomado de [48].

3.1.3. EnhanceNet

La red EnhanceNet es una red GAN diseñada para una tarea llamada súper resolución, que consiste en a partir de una imagen pequeña aumentarla de resolución sin que *se vea pixelada* [54]. En este Trabajo Fin de Grado se utiliza esta red debido a las limitaciones que tiene CycleGAN con la resolución: con esta red podemos utilizar nuestra CycleGAN y posteriormente ampliar la resolución drásticamente para ofrecer imágenes de mayor calidad.

EnhanceNet es una de las muchas redes que realizan esta tarea, pero la hemos elegido en este Trabajo Fin de Grado debido a que los autores tienen



Figura 3.6: Ejemplo de EnhanceNet. Tomado de [54].

publicados los pesos de la red ya entrenada, lo que nos ahorra entrenar otra red desde 0. Asimismo esto nos permite tener un subsistema que amplíe la resolución de calidad y a coste 0, prescindiendo de servicios de pago.

Output size	Layer
$w \times h \times c$	Input I_{LR}
	Conv, ReLU
$w \times h \times 64$	Residual: Conv, ReLU, Conv
	...
$2w \times 2h \times 64$	2x nearest neighbor upsampling Conv, ReLU
$4w \times 4h \times 64$	2x nearest neighbor upsampling Conv, ReLU Conv, ReLU
$4w \times 4h \times c$	Conv Residual image I_{res}
	Output $I_{est} = I_{bicubic} + I_{res}$

Figura 3.7: Arquitectura de EnhanceNet. Tomado de [54].

Si bien no es necesario centrarnos en la red (en este trabajo se utiliza la red como una caja negra), en la figura 3.7 se describe la arquitectura de la red. Viendo la imagen vemos muchos nombres que nos resultan familiares: tenemos que EnhanceNet es una red convolucional que mediante bloques residuales aprende a ampliar las imágenes. Más concretamente, son 10 bloques residuales, lo que ayuda a que el modelo aprenda rápido [55]; además de utilizar una función de error (dentro de las cuatro que posee) inspirada en las redes CycleGAN.

Por último, la red ofrece una *imagen residual*, que es la diferencia entre la imagen deseada y la ampliación mediante un algoritmo conocido como interpolación bicúbica, el cual se detalla en [56].

3.2. Datos utilizados para elaborar el sistema

Los datasets utilizados en este Trabajo Fin de Grado son tres: *monet2photo*, *vangogh2photo* y *cezanne2photo*. Todos los datasets tienen la misma estructura: por un lado tenemos los cuadros del artista, mientras que por otro disponemos un conjunto de fotografías de paisajes.

Dichos datasets fueron creados por los autores del paper del modelo CycleGAN [49]. De acuerdo con sus especificaciones, los autores descargaron las imágenes de los cuadros de la web wikiart.org, descartando los bocetos y las obras obscenas. Asimismo, las fotos fueron descargadas de flickr.com buscando por las etiquetas *landscape* y *landscapephotography*, obviando fotografías en blanco en negro. Todas las imágenes fueron redimensionadas a 256x256 pixeles.

En total, se disponen de 1074 imágenes de Monet, 584 de Cézanne y 401 de Van Gogh, ya que los cuadros de Monet fueron filtrados para incluir únicamente cuadros de paisajes y sólo se incluyeron los trabajos finales de Van Gogh, buscando representar su estilo artístico más reconocible.

Por otra parte, la estructura del cada dataset es la siguiente: se disponen de cuatro carpetas, llamadas *trainA*, *trainB*, *testA* y *testB*. Las carpetas con sufijo A contienen los cuadros, mientras que los que tienen sufijo B las fotografías. En las figuras 3.8a y 3.8b mostramos una parte las carpetas *trainA*, *trainB* del dataset *cezanne2photo*.

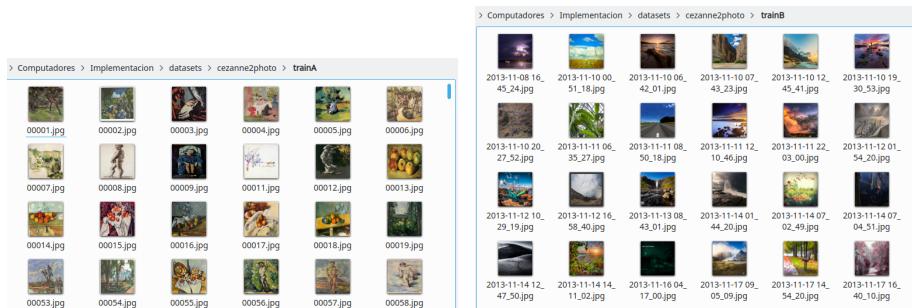


Figura 3.8: Vista previa del dataset *cezanne2photo*

Por último, los autores crearon más datasets para su nueva arquitectura de red neuronal. Uno de sus autores, Taesung Park, los publicó en la url https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/, dirección que utilizaremos en la implementación para descargarlos automáticamente. Asimismo, TensorFlow, en su herramienta de datasets, también los recoge para su uso.

Index of /~taesung_park/CycleGAN/datasets

Name	Last modified	Size	Description
Parent Directory	-	-	-
NOTICE	2019-08-12 20:45	227	
ae_photos.zip	2017-04-03 22:06	10M	
apple2orange.zip	2017-03-28 13:51	75M	
cezanne2photo.zip	2017-03-28 13:51	267M	
cityscapes.zip	2019-08-12 20:45	325	
facades.zip	2017-03-29 23:23	34M	
grumpifycat.zip	2020-08-03 20:58	19M	
horse2zebra.zip	2017-03-28 13:51	111M	
iphone2dslr_flower.zip	2017-03-30 12:05	324M	
maps.zip	2017-03-26 19:17	1.4G	
mini.zip	2018-06-07 16:05	1.8M	
mini_colorization.ta_>	2019-01-01 16:38	303K	
mini_colorization.zip	2019-01-01 16:44	304K	
mini_pix2pix.zip	2018-06-07 16:08	1.5M	
monet2photo.zip	2017-03-26 19:17	291M	
summer2winter_yosemite.>	2017-03-26 19:17	126M	
ukiyoe2photo.zip	2017-03-26 19:17	279M	
vangogh2photo.zip	2017-03-26 19:17	292M	

Figura 3.9: Datasets creados por el equipo de Cycle GAN

3.3. Tecnologías utilizadas

Dentro del mundo del Machine Learning existen numerosas herramientas de trabajo, siendo un entorno que se renueva a una velocidad de vértigo.

El "verano" que atraviesa la Inteligencia Artificial en general y la Visión Artificial en particular provocan que numerosas instituciones, empresas y particulares apuesten por desarrollar nuevas herramientas o mejorar las ya existentes. Algunos de esos contribuyentes son actores de la industria tan destacados como Google, Facebook y NVIDIA.

Por otra parte, en este Trabajo Final de Grado se han utilizado herramientas principalmente de código abierto y con amplia documentación, las cuales se detallan a continuación.

3.3.1. Python

Python es un lenguaje de programación que apareció en 1991, diseñado por Guido Van Rossum. Actualmente administrado por la Python Software Foundation y distribuido mediante una licencia de código abierto denominada Python Software Foundation License.

En los últimos años ha ganado mucho peso en todos los ámbitos de la Informática, debido a su filosofía centrada en la legibilidad del código y su facilidad de aprendizaje. Sus características principales son las siguientes:

- Soporta programación estructurada, orientada a objetos, imperativa y funcional, sin forzar al usuario a decantarse por ninguno de ellos y pudiendo cambiar entre ellos en cualquier momento. Este enfoque aúna las ventajas de cada paradigma, facilitando a los programadores utilizar la solución que deseen sin verse limitados por el lenguaje de programación.

- Posee un diseño que facilita la extensión del mismo y la interoperabilidad con otros lenguajes de programación, estimulando la creación de librerías y módulos compatibles con Python.
- A diferencia de otros lenguajes, su código es interpretado por un intérprete. Este enfoque proporciona un mayor soporte a la multiplataforma, existiendo implementaciones de Python escritas en lenguajes como C, Java o C# para una gran cantidad de sistemas operativos.
- Proporciona un modo interactivo similar a una terminal de comandos, lo que permite un desarrollo más cómodo para el programador al probar bloques de código independientemente de forma sencilla.
- Su sistema de tipos es fuertemente tipado y dinámico, lo que aporta una mayor seguridad en las operaciones del programa sin renunciar a facilitar el desarrollo y el mantenimiento del mismo.
- Incorpora soporte para la instalación de módulos mediante la herramienta pip, utilizada en este Trabajo Final de Grado para facilitar la instalación de las dependencias del sistema implementado.

3.3.2. TensorFlow

TensorFlow es una librería de cálculo numérico orientada a la computación distribuida, creada por Google y bajo el amparo de la licencia de código abierto Apache 2.0 desde noviembre de 2015.

Ha ganado mucho protagonismo en los últimos años ya que principalmente facilita el uso de tarjetas gráficas para acelerar la computación que necesitan los modelos de Machine Learning. Sus principales características son:

- Permite variar el hardware de forma transparente al programador, sin tener que realizar código adicional o transformarlo para las diferentes implementaciones. En otras palabras, implementa el Principio de inversión de dependencias, logrando que el programador dependa únicamente de abstracciones y no de los detalles de implementación.
- Internamente utiliza grafos de flujo de datos, lo que permite ciertas optimizaciones y un procesado vectorial de los propios datos, aumentando la eficiencia del software. Con la publicación de TensorFlow 2.0 el 30 de septiembre de 2019, no es necesario programar forzosamente mediante dichos grafos, permitiendo el uso de un paradigma imperativo (*Eager Execution*) más intuitivo de cara al programador.
- Proporciona API para los siguientes lenguajes: Python (utilizada en este Trabajo Final de Grado), C++, Haskell, Java, Go y Rust.

3.3.3. Tensorboard

Tensorboard es la interfaz gráfica incluida con TensorFlow para facilitar la comprensión del software implementado. Posee herramientas para la visualización de gráficas asociadas a las métricas del modelo, lo que permite

evaluar de forma rápida el proceso de aprendizaje del modelo ya que indica, además de la métrica, los metadatos que queramos además del tiempo transcurrido de ejecución.

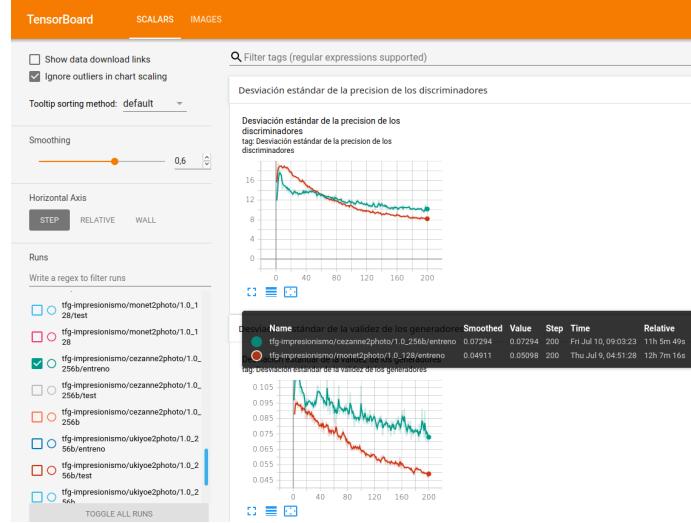


Figura 3.10: Ejemplo de las gráficas proporcionadas por Tensorboard

Asimismo dispone de opciones para visualizar imágenes relacionadas con el modelo, el grafo de computación ejecutado por TensorFlow, histogramas y distribuciones.

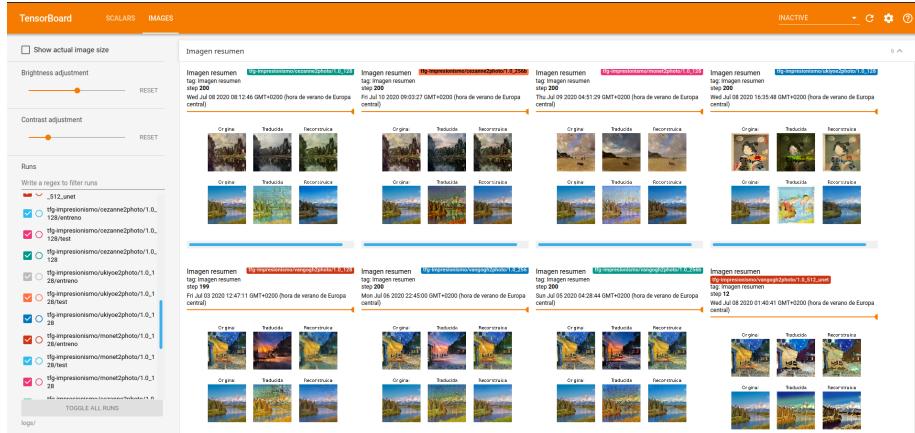


Figura 3.11: Ejemplo de la visualización de fotos mediante Tensorboard

3.3.4. Keras

Keras es una API de Redes Neuronales que permite un desarrollo y mantenimiento cómodo y sencillo de las mismas. Desarrollada por François

Chollet, fue lanzada el 27 de marzo de 2015 bajo la licencia de código abierto MIT. Dicha interfaz puede funcionar bajo TensorFlow, Theano y Microsoft Cognitive Toolkit, abstrayendo el modelo del *background* computacional.

La combinación de TensorFlow con Keras es ampliamente utilizada en el sector debido a su sencillez de uso, potencia y eficacia. De hecho, TensorFlow en su versión 2 renunció a sus API de modelos predefinidos para ofrecer una integración completa y óptima con Keras.

3.3.5. CUDA

CUDA son las siglas de *Compute Unified Device Architecture* (Arquitectura Unificada de Dispositivos de Cómputo), una plataforma para realizar procesamiento paralelo mediante las GPU de NVIDIA, creada en el año 2007. Esta herramienta permite utilizar las ventajas del paralelismo masivo que ofrecen las GPU mediante un dialecto de C, si bien existen enlaces para poder usarla desde Python, Fortran y MATLAB.

Con esta tecnología, podemos paralelizar de forma sencilla nuestras aplicaciones. Así mismo dispone de una extensión, llamada cuDNN, que optimiza las aplicaciones de Redes Neuronales, reduciendo aún más los tiempos de cómputo. De hecho, en este Trabajo Final de Grado es utilizada indirectamente, ya que es la plataforma que usa TensorFlow para paralelizar sus operaciones; si bien requiere una configuración un tanto problemática. Dicha situación se detallará en la sección 3.5.1

3.3.6. Google Cloud Platform

Google Cloud Platform es la plataforma de computación en la nube propiedad de Google, en la que se ofertan todo tipo de servicios. Mediante esta herramienta, los usuarios pueden contratar servicios de forma rápida y flexible, sujetos únicamente al pago por uso de los mismos. Entre otros, provee servicios de Inteligencia Artificial, Big Data y almacenamiento, además de máquinas virtuales totalmente configurables y de elevado rendimiento.

En este Trabajo Final de Grado se ha utilizado esta plataforma para acceder a las GPU y TPU de Google, las cuales permiten acelerar en gran medida los tiempos de entrenamiento del sistema propuesto.

3.3.7. Flask

Flask es un *microframework* elaborado para desarrollo web mediante Python, creado por Armin Ronacher y liberado bajo la licencia de código abierto BSD. A diferencia de otros frameworks, Flask está pensado para crear aplicaciones web rápidamente, con una curva de aprendizaje suave para el programador; lo que permite desarrollar servicios de forma ágil y rápida.

Además añade soporte para la creación de API REST de forma extremadamente sencilla, que es el uso que le daremos en este Trabajo Final de Grado.

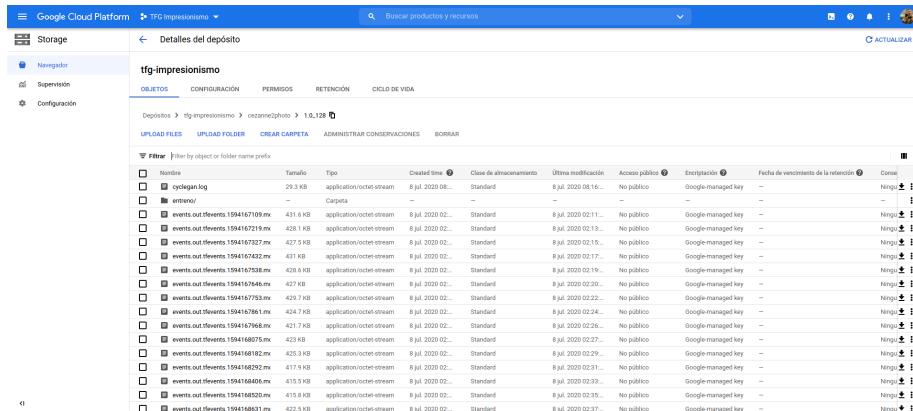


Figura 3.12: Ejemplo de visualización de *logs* de TensorBoard en el servicio de almacenamiento de Google Cloud Plataform

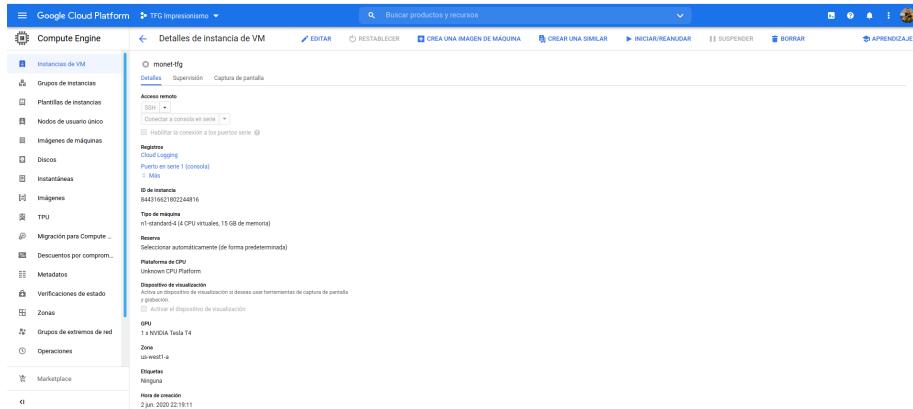


Figura 3.13: Visualización de la descripción de la máquina virtual utilizada en este TFG alojada en Google Cloud Plataform

3.3.8. Git y GitHub

Git es un sistema de control de versiones *open source* diseñado por Linus Torvalds (conocido por ser el artífice de Linux). Este sistema de control de versiones se caracteriza por su concepción distribuida, lo que permite prescindir de un servidor si así lo deseamos. Diseñado para el desarrollo no lineal de proyectos, fue desarrollado para la gestión del núcleo de Linux, lo que junto a su velocidad, eficiencia y facilidad de uso lo ha convertido en uno de los sistemas más utilizados en todo el mundo.

Por otra parte, GitHub es una plataforma propietaria diseñada principalmente para el alojamiento remoto de repositorios Git. Si bien no es necesaria para utilizar Git, ha adoptado una inmensa popularidad debido a que nos permite tener repositorios privados (ahormando a los desarrolladores el despliegue y mantenimiento de un servidor) sin renunciar a crear repositorios públicos con los que compartir nuestro código hacia la comunidad. En este

Trabajo Fin de Grado se ha utilizado tanto Git como GitHub para el control de versiones de la implementación *software*.

3.4. Diseño general del sistema

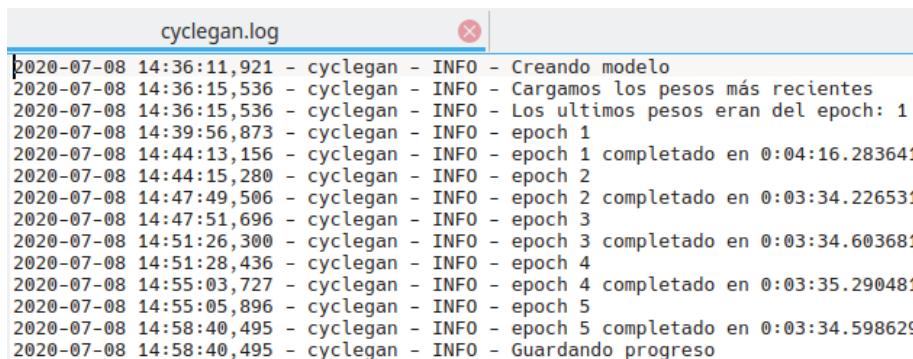
3.4.1. Sistema principal

Este sistema es el núcleo del proyecto, ya que es el sistema que implementa la red Cycle GAN. En primer lugar expondremos la estructura de archivos del mismo, para posteriormente explicar las opciones de configuración, los casos de uso del sistema y por último la funcionalidad de todos los ficheros de código fuente.

3.4.1.1. Estructura de archivos del sistema

Se disponen de los siguientes directorios (además del archivo requirements.txt que es utilizado para la instalación de dependencias):

- **configuracion:** contiene los archivos de configuración .json.
- **datasets:** recoge los conjuntos de imágenes a utilizar por el sistema (organizados por su nombre) para su entrenamiento y para la generación del conjunto test de resultados.
- **input:** dispone de las fotos que el usuario desee transformar a cuadros.
- **logs:** contiene los logs del sistema (según dataset y versión del sistema), tanto como los de depuración como los de tensorboard.



```
cyclegan.log
2020-07-08 14:36:11,921 - cyclegan - INFO - Creando modelo
2020-07-08 14:36:15,536 - cyclegan - INFO - Cargamos los pesos más recientes
2020-07-08 14:36:15,536 - cyclegan - INFO - Los ultimos pesos eran del epoch: 1
2020-07-08 14:39:56,873 - cyclegan - INFO - epoch 1
2020-07-08 14:44:13,156 - cyclegan - INFO - epoch 1 completado en 0:04:16.283641
2020-07-08 14:44:15,280 - cyclegan - INFO - epoch 2
2020-07-08 14:47:49,506 - cyclegan - INFO - epoch 2 completado en 0:03:34.226531
2020-07-08 14:47:51,696 - cyclegan - INFO - epoch 3
2020-07-08 14:51:26,300 - cyclegan - INFO - epoch 3 completado en 0:03:34.603681
2020-07-08 14:51:28,436 - cyclegan - INFO - epoch 4
2020-07-08 14:55:03,727 - cyclegan - INFO - epoch 4 completado en 0:03:35.290481
2020-07-08 14:55:05,896 - cyclegan - INFO - epoch 5
2020-07-08 14:58:40,495 - cyclegan - INFO - epoch 5 completado en 0:03:34.598629
2020-07-08 14:58:40,495 - cyclegan - INFO - Guardando progreso
```

Figura 3.14: Muestra de logs del sistema

- **modelos:** contiene los modelos que utiliza el sistema ya entrenados, organizados por dataset y versión. Dispone además de los *checkpoints* de los propios modelos, guardados cada 5 epochs.
- **output:** recoge los resultados del sistema, organizados nuevamente por versión y dataset. Para cada uno de ellos dispone de las imágenes del usuario en la raíz, así como dos subdirectorios llamados *test_foto* y *test_pintor*, los cuales guardan los resultados finales sobre los conjuntos de test del dataset.

- src: dispone de los ficheros de código fuente.

3.4.1.2. Configuración del sistema

Tenemos dos secciones de configuración: los archivos .json y los parámetros de las lanzaderas del modelo.

3.4.1.2.1 Archivos de configuración .json

Son leídos por el sistema al arrancar. Dispone de las siguientes secciones:

- configuración.modelo: ajusta los parámetros del sistema, tales como la tasa de aprendizaje, las dimensiones de las imágenes, los epochs del entrenamiento y los filtros de los generadores y discriminadores, entre otras opciones.
- url: dispone de la url de la API del sistema ampliación de resolución (puede ser local o en línea) y de la url datasets, desde donde el sistema descarga los datasets en caso de no disponer de ellos.
- gcp: contiene la ruta del contenedor de Google Cloud Platform donde almacenar los logs del sistema si se entrena en dicha plataforma.
- dataset: guarda, para cada dataset, las dos imágenes que se usan en la imagen resumen mostrada en tensorboard para seguir de forma gráfica el entrenamiento del sistema.
- varios: dispone de una opción, *mascara_logs*, que como su nombre indica es la máscara usada en el formateo de los logs del sistema.

La figura 3.15 indica un fichero json de ejemplo. El fichero que utilizemos debe contener, como mínimo, todos los campos mostrados de todas las secciones salvo de *dataset*, en la que sólo se debe disponer de al menos un registro con los campos *imagen_pintor_muestra* y *imagen_foto_muestra*.

3.4.1.2.2 Opciones de las lanzaderas

Son las opciones relacionadas los ficheros lanzadera, explicados en las secciones 3.4.1.3.1 y 3.4.1.3.2. Dichas opciones se ajustan al invocar al sistema mediante la linea de comandos (o en las configuraciones si utilizamos un IDE), y son:

- versión (indicada con -v o –version): Especifica el nombre de la versión con la que se guardarán los archivos.
- dataset (expresada con -d o –dataset): Especifica el dataset a utilizar
- configuracion (indicada con -c o –configuracion): Indica el fichero de configuración .json a utilizar. Debe indicarse la extensión .json en el uso del mismo.
- arquitectura (expresada con -a o –arquitectura): Exclusiva de la lanzadera de entrenamiento, indica al sistema qué arquitectura del generador se debe de usar: u-net (unet) o ResNet (resnet).

```
{
  "configuracion_modelo":
  {
    "tasa_aprendizaje": "0.0002",
    "lambda_reconstruccion": "10.0",
    "lambda_validacion": "1",
    "lambda_identidad": "2",
    "ancho": "256",
    "alto": "256",
    "canales": "3",
    "epochs": "200",
    "tamanio_buffer": "1000",
    "tamanio_batch": "1",
    "filtros_generador": "64",
    "filtros_discriminador": "64"
  },
  "url":
  {
    "api_aumento": "http://localhost:5000/TFG-Computadores/api-aumento/aumento",
    "datasets": "https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/"
  },
  "gcp":
  {
    "bucket": "gs://tf2-impressionismo/"
  },
  "dataset":
  {
    "monet2photo":
    {
      "imagen_pintor_muestra": "00360.jpg",
      "imagen_foto_muestra": "2014-08-24 16_41_27.jpg"
    },
    "cezanne2photo":
    {
      "imagen_pintor_muestra": "00100.jpg",
      "imagen_foto_muestra": "2014-08-24 16_41_27.jpg"
    },
    "vangogh2photo":
    {
      "imagen_pintor_muestra": "00021.jpg",
      "imagen_foto_muestra": "2014-08-24 16_41_27.jpg"
    }
  },
  "varios":
  {
    "mascara_logs" : "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
  }
}
```

Figura 3.15: Ejemplo de archivo de configuración json

3.4.1.3. Casos de uso

Este sistema dispone de dos casos de uso, disponibles desde dos puntos de arranque (una para cada caso), llamadas lanzaderas, siendo una para el entrenamiento del sistema y otra para su puesta en marcha en la composición de cuadros (o producción).

3.4.1.3.1 Entreno del sistema

Si deseamos entrenar al sistema, debemos de ejecutar la lanzadera *lanzadera_entreno* con las opciones deseadas. Se asegura de que el fichero de configuración exista, que la arquitectura sea válida y que el dataset exista

(descargándolo y extrayéndolo si no se dispone del mismo), para posteriormente crear la red, cargar el dataset y realizar el entrenamiento. Una vez dicho entrenamiento finaliza, serializa la red y finaliza la ejecución.

3.4.1.3.2 Creación de imágenes

Sucede cuando invocamos la lanzadera *lanzadera_produccion*. Al igual que el caso anterior, comprueba que el fichero de configuración y el dataset existan, para posteriormente pintar las fotos indicadas en output y en datasets\nombre correspondiente\testB y *fotografiar* los cuadros disponibles en datasets\nombre correspondiente\testA; ignorando los resultados ya existentes.

3.4.1.4. Ficheros de código fuente

En esta sección se explica únicamente las labores que realiza cada módulo de código, no cada función concreta. Para observar las funciones en detalle, remitimos al anexo B.1

3.4.1.4.1 ampliar_imagen

Como el lector habrá supuesto, se encarga de ampliar la imagen que recibe de entrada. Esta función utiliza el sistema ampliador de resolución si está disponible (a través de una petición POST a la API del sistema, pasando la imagen como un string base64 y recibiendo un array de bytes), y en caso contrario amplia la imagen en un factor de aumento (recibido por parámetro).

3.4.1.4.2 capas_extra

Un módulo muy liviano, únicamente implementa una capa especial mediante una clase para la arquitectura resnet, conocida como *ReflectionPadding2D*.

3.4.1.4.3 cargador_imagenes

Define la clase CargadorImagenes, la cual alimenta de imágenes a la red en el proceso de entrenamiento de la misma encargándose de la lectura de las imágenes y su preprocesamiento. Asimismo, calcula los n_batches que van a ejecutarse durante el entrenamiento.

3.4.1.4.4 cyclegan

Esta es la clase principal del sistema. Contiene la implementación de toda la red, incluyendo los generadores unet y resnet. Se encarga de crear la red, entrenarla, cargar el último punto de guardado del entrenamiento si existe y de guardar la red cuando finaliza el entrenamiento para que pueda ser utilizada en la creación de cuadros. Por último, realiza la inferencia de imágenes (o transformación de cuadro a imagen y viceversa).

3.4.1.4.5 lanzadera_entreno

Arranca el sistema para el entrenamiento de la red, funcionalidad descrita en la sección 3.4.1.3.1.

3.4.1.4.6 lanzadera_produccion

Análogamente al caso anterior, inicializa el sistema para la inferencia de imágenes, funcionalidad descrita en la sección 3.4.1.3.2.

3.4.1.4.7 procesado_imagenes

Dispone de funciones relacionadas con el procesado y transformaciones (tanto de tipos como de resolución) de imagen.

3.4.1.4.8 singleton

Es un módulo muy sencillo, el cual contiene la definición del patrón de diseño singleton, el cual es usado por las clases CargadorImagenes y Utilidades.

3.4.1.4.9 utilidades

Almacena todas las funciones útiles para el resto de clases: conversión de rutas de fichero, marcas de tiempo, existencia de *gsutil* (herramienta de Google Cloud Platform para la gestión de sus contenedores de almacenamiento), copia de logs al contenedor de Google Cloud Platform, almacenamiento de rutas de ficheros, lectura del fichero de configuración...

3.4.2. Sistema ampliador de resolución

Este sistema, como su propio nombre indica, se encarga de realizar la ampliación de la resolución (en un factor de 4) utilizando la red GAN EnhanceNet. La implementación realizada en este Trabajo Fin de Grado se ha basado en la implementación oficial realizada por los propios autores en TensorFlow 1, adaptándola para su uso mediante una API REST.

No obstante, se ha modificado el código de forma que puedan deshacerse los cambios y utilizarse de forma análoga al sistema principal. Esa versión alternativa se recoge en el repositorio de GitHub <https://github.com/VicDominguez/EnhanceNet-TFG/tree/91058eba0ebbcd1f126d441ec15ef3987a3caa5f>.

Pasemos ahora a explicar el diseño de este sistema. A diferencia del anterior, este sistema no dispone de estructura de ficheros reseñable (únicamente contiene los ficheros de código, el archivo con los pesos de la red y el fichero requirements.txt de dependencias), ni de opciones de configuración.

Asimismo, sólo disponemos de un caso de uso: recibir peticiones POST con una imagen adjunta (codificada en base64), realizar el aumento de resolución sobre la misma y devolver la imagen ampliada al cliente.

Por último, expliquemos la funcionalidad de cada módulo de código fuente:

3.4.2.1. controlador_modelo

Sirve de adaptador entre el modelo y las posibles ejecuciones del sistema (API REST, fichero imagen y fichero base64). En la implementación utilizada

en este Trabajo Fin de Grado únicamente se utilizan las funciones relacionadas con la API REST.

3.4.2.2. modelo

Contiene la definición de la red EnhanceNet, así como la función inferencia, la cual crea la red, carga los pesos de la misma y ejecuta la ampliación de la misma.

3.4.2.3. principal

Despliega la API REST utilizando Flask. Se encarga de recibir la imagen a ampliar mediante peticiones POST, además de comprobar la integridad y el tipo de imagen.

3.4.2.4. procesado

Dispone de funciones para el procesamiento de la entrada y de la salida (tanto de fichero como de la API), además de encargarse del preprocesamiento de la imagen.

3.4.2.5. utils

Contiene funciones relacionadas con la gestión de rutas y constantes necesarias para el sistema.

Si se desea consultar el código de la implementación, remitimos al anexo B.3.

3.5. Guía de uso

3.5.1. Instalación de dependencias

Para ejecutar el código creado en este Trabajo Fin de Grado tenemos que satisfacer ciertas dependencias. En los primeros pasos las elecciones de versiones no son caprichosas, ya que este Trabajo Fin de Grado se ha programado principalmente bajo la versión 2.2.1 de TensorFlow (no impidiendo la ejecución de la versión 1.14 también utilizada). La figura 3.16 muestra las versiones compatibles entre sí.

Enumeremos el proceso de instalación de las mismas:

1. Disponer de una tarjeta gráfica nVidia en nuestro equipo que soporte (y si no lo tenemos, instalarlo) como mínimo el driver 418.39 de la propia nVidia para poder instalar una versión CUDA igual o superior a 10.1. Se recomienda que disponga de al menos 2GB de VRAM para realizar la creación de cuadros.
2. Instalaremos como mínimo CUDA 10.1 siguiendo el tutorial oficial de nVidia: <https://docs.nvidia.com/cuda/archive/10.1/index.html>
3. Tras ello, instalamos la versión 7.6 para la versión de CUDA que hayamos instalado, siguiendo nuevamente el tutorial oficial <https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html>

Version	Python version	Compiler	Build tools	cuDNN	CUDA
tensorflow-2.2.0	3.5-3.8	GCC 7.3.1	Bazel 2.0.0	7.6	10.1
tensorflow-2.1.0	2.7, 3.5-3.7	GCC 7.3.1	Bazel 0.27.1	7.6	10.1
tensorflow-2.0.0	2.7, 3.3-3.7	GCC 7.3.1	Bazel 0.26.1	7.4	10.0
tensorflow_gpu-1.14.0	2.7, 3.3-3.7	GCC 4.8	Bazel 0.24.1	7.4	10.0
tensorflow_gpu-1.13.1	2.7, 3.3-3.7	GCC 4.8	Bazel 0.19.2	7.4	10.0
tensorflow_gpu-1.12.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.11.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.10.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.9.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.11.0	7	9
tensorflow_gpu-1.8.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.10.0	7	9
tensorflow_gpu-1.7.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.9.0	7	9
tensorflow_gpu-1.6.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.9.0	7	9
tensorflow_gpu-1.5.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.8.0	7	9
tensorflow_gpu-1.4.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.5.4	6	8
tensorflow_gpu-1.3.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.5	6	8
tensorflow_gpu-1.2.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.5	5.1	8
tensorflow_gpu-1.1.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.2	5.1	8
tensorflow_gpu-1.0.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.4.2	5.1	8

Figura 3.16: Requisitos de versiones de nVidia de TensorFlow. Extraído de [57]

4. Si aún no lo tenemos instalado, descargamos el intérprete de Python y lo instalamos. Se recomienda utilizar un Entorno Integrado de Desarrollo (o IDE) como PyCharm.
5. Si queremos ejecutar el sistema principal, necesitaremos tener instalado Git ya que es obligatorio para instalar dos dependencias concretas. Se puede instalar siguiendo la siguiente ayuda: <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git>
6. Instalamos el gestor de paquetes de Python pip siguiendo nuevamente el tutorial oficial: <https://pip.pypa.io/en/stable/installing/>
7. Por último, según el subsistema que queramos ejecutar necesitaremos instalar las dependencias concretas del mismo. Si utilizamos un IDE como PyCharm el propio programa nos ayuda a instalarlas. Si no es ese caso, podemos instalarlas mediante el siguiente comando:

```
pip3 install -r requirements.txt
```

3.5.2. Creación de cuadros

La creación de cuadros es extremadamente sencilla: en la carpeta input situaremos nuestras imágenes, como se ilustra en la figura 3.17.

Tras ello, ejecutamos el sistema mediante el acceso *lanzadera_producción* (y con el subsistema de ampliación ejecutándose si así lo deseamos). Accedemos a la

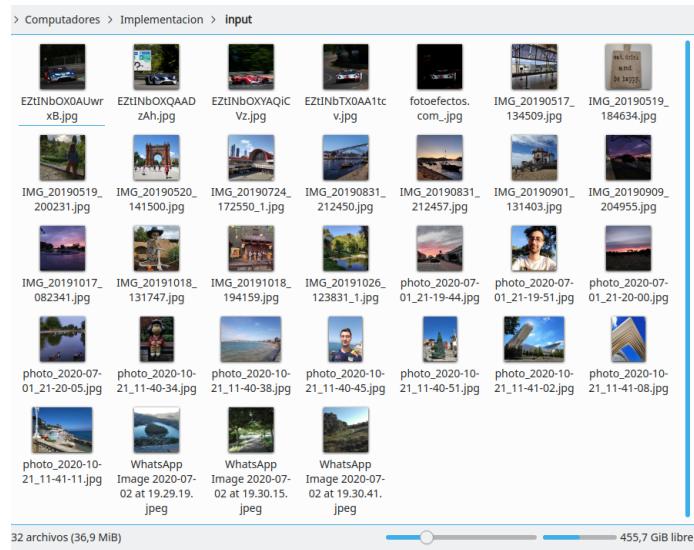


Figura 3.17: Ejemplo de contenido input

carpeta output >nombre de nuestro dataset >nombre de nuestra versión (campo -v de *lanzadera_producción*) y ahí están nuestros cuadros, tal como refleja la figura 3.18

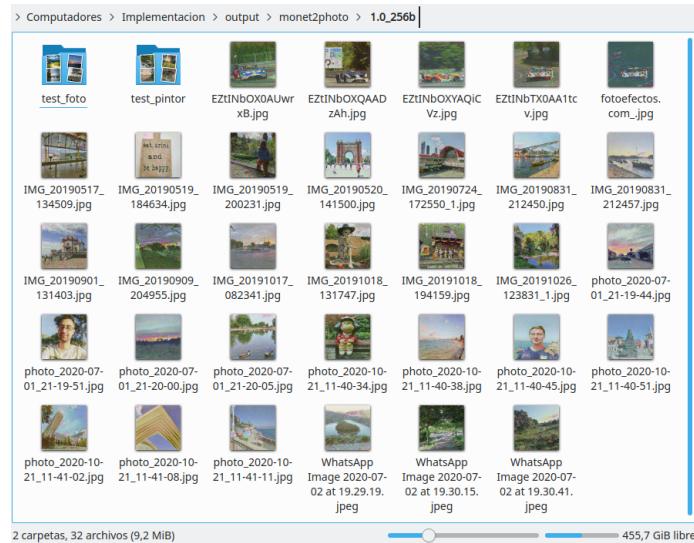


Figura 3.18: Ejemplo de cuadros generados

3.5.3. Creación de modelos personalizados

En este aspecto tenemos que distinguir varios casos:

- Si queremos modificar los parámetros del modelo, únicamente tenemos

que crear o modificar un fichero .json de configuración y editar los valores deseados de los campos de la sección *configuracion_modelo*. No olvidar indicar a la lanzadera el fichero de configuración a utilizar.

- Si deseamos entrenar al sistema con otros datos, tenemos dos supuestos:

1. Si queremos utilizar un dataset recogido en la figura 3.9, tenemos que indicarlo en el campo *-d* de la lanzadera y recoger en el fichero .json de configuración una entrada similar a la indicada en la sección *dataset*; como se muestra en la figura 3.19.

```
{
  "configuracion_modelo": {
    "tasa_aprendizaje": "0.0002"...
  },
  "url": {
    "api_aumento": "http://localhost:5000/TFG-Computadores/api-aumento/aumento"...
  },
  "gcp": {
    "bucket": "gs://tfg-impressionismo/"...
  },
  "dataset": {
    {
      "apple2orange": {
        {
          "imagen_pintor_muestra": "n07740461_10750.jpg",
          "imagen_foto_muestra": "n07749192_1010.jpg"
        }
      }
    }
  }
}
```

Figura 3.19: Parámetros para un nuevo dataset

2. Si queremos crear un nuevo dataset (por ejemplo, uniendo los datasets monet2photo, cezanne2photo y vangogh2photo), además de realizar el procedimiento del caso anterior, a la hora de crear el dataset tenemos que respetar la estructura en carpetas *trainA*, *trainB*, *testA* y *testB* (todas ellas guardadas en el mismo directorio). Posteriormente debemos bien guardar dichas carpetas en el directorio datasets del sistema o comprimir dichas carpetas en un único archivo de extensión zip y subirlas a una url. Dicha dirección se la indicamos al archivo de configuración en el campo *datasets* de la sección *url*, como se indica en la ilustración 3.20.

```
{
  "configuracion_modelo": {
    "tasa_aprendizaje": "0.0002"...
  },
  "url": {
    {
      "api_aumento": "http://localhost:5000/TFG-Computadores/api-aumento/aumento",
      "datasets": "https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/"
    },
    "gcp": {
      "bucket": "gs://tfg-impressionismo/"...
    },
    "dataset": {
      {...}
    },
    "varios": {
      {"mascara_logs": "%(asctime)s - %(name)s - %(levelname)s - %(message)s"...
    }
  }
}
```

Figura 3.20: Parámetro url para un nuevo dataset

Capítulo 4

Resultados

4.1. Resultados obtenidos

Los resultados aquí presentados se han realizado utilizando el sistema ampliador de resolución (EnhanceNet) y con las versiones (del sistema principal) 1.0_256b en el caso de Monet y Cézanne, y 1.0_256 en el caso de Van Gogh. Dichas versiones implementan la arquitectura resnet y utilizan el archivo de configuración *configuracion_256.json* suministrado en el anexo B.2.

4.1.1. Paso de fotografías a cuadros

4.1.1.1. Paisajes del dataset

4.1.1.1.1 Cézanne

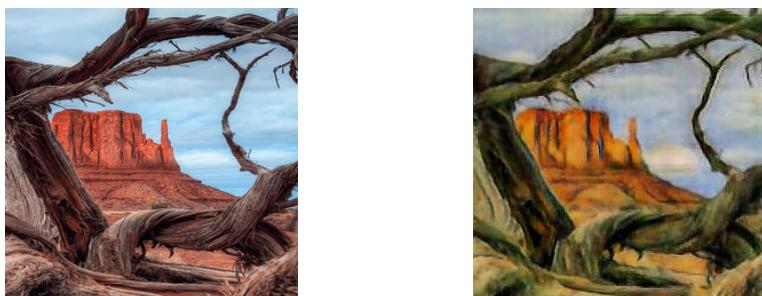


Figura 4.1: Cuadro de Cézanne generado en el desierto

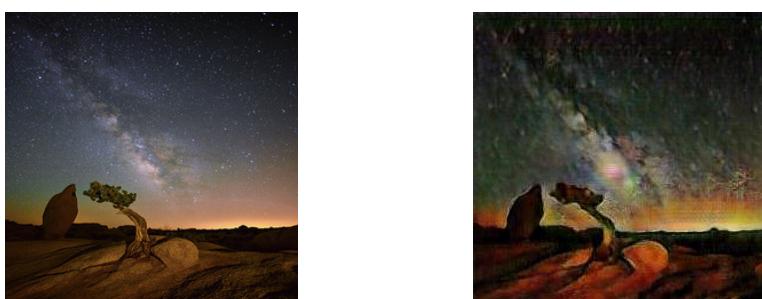


Figura 4.2: Cuadro de Cézanne generado en la noche

En este caso puede verse a simple vista cómo ha aprendido el sistema la rojiza paleta de colores característica de Cézanne, además de los cielos. En el caso de la figura 4.2 puede verse como la parte de luz ha sido exagerada por el modelo, debido a que no ha recibido apenas entrenamiento en imágenes nocturnas.



Figura 4.3: Cuadro de Cézanne generado en una orilla



Figura 4.4: Cuadro de Cézanne generado de unos edificios

4.1.1.1.2 Monet

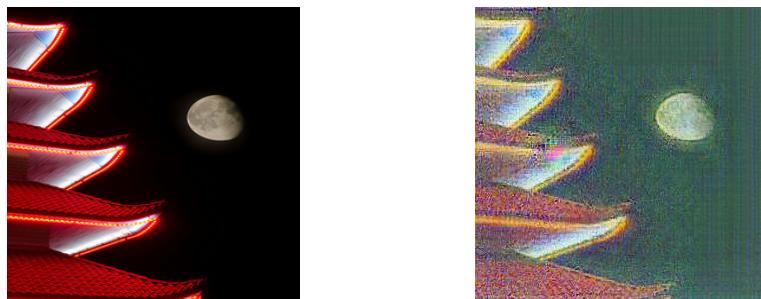


Figura 4.5: Cuadro de Monet generado a partir de un edificio japonés de noche

Podemos ver a simple vista que el modelo ha interpretado los cuadros de Monet como aquellos con tonalidades azules frías. Es especialmente llamativo el caso de la figura 4.5: el sistema *blanquea* el cielo de la noche, nuevamente debido a la falta de ejemplos de cuadros nocturnos de Monet.



Figura 4.6: Cuadro de Monet generado a partir de un puente



Figura 4.7: Cuadro de Monet generado a partir de edificaciones en el agua



Figura 4.8: Cuadro de Monet generado a partir de un puente de madera

4.1.1.3 Van Gogh

En este caso pueden verse más diferencias más allá de las paletas de color: se ve fácilmente las pinceladas bruscas características de Van Gogh, dando una textura especial a los cuadros.



Figura 4.9: Cuadro de Van Gogh generado a partir de un campo con bolas



Figura 4.10: Cuadro de Van Gogh generado a partir de una casa en el mar



Figura 4.11: Cuadro de Van Gogh generado a partir de una montaña



Figura 4.12: Cuadro de Van Gogh a partir de un ferrocarril en el agua

4.1.1.2. Fotos del autor**4.1.1.2.1 Cézanne**

Figura 4.13: Cuadro de Cézanne generado en el arco del triunfo de Barcelona

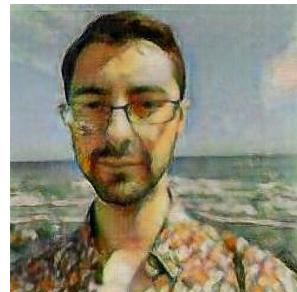


Figura 4.14: Cuadro de Cézanne generado a partir de una foto en una playa del autor

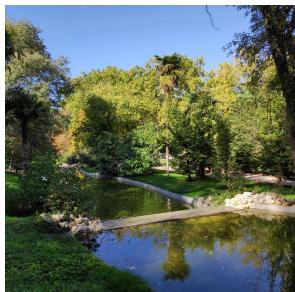


Figura 4.15: Cuadro de Cézanne generado de una foto de un parque de Madrid

En la figura 4.14 puede verse el cambio de textura en la camisa, simplificándola; también puede apreciarse en la ilustración 4.13. Por otra parte es especialmente notable cómo cambian las texturas de los árboles y sus colores en las figuras 4.16 y 4.15, señal de que el sistema las ha aprendido con soltura.



Figura 4.16: Cuadro de Cézanne generado de una foto en el puerto de Honduras (Cáceres)



Figura 4.17: Cuadro de Cézanne generado de una Hyosung GTR 650

4.1.1.2.2 Monet

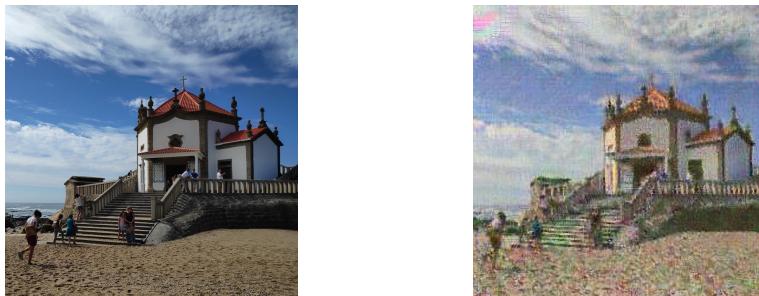


Figura 4.18: Cuadro de Monet generado a partir de una iglesia costera portuguesa

Se mantiene el impacto reducido de las transformaciones del modelo, señal de que el modelo quizás haya aprendido poco. Puede verse que los elementos que más mutan son el cielo y las estructuras metálicas de la ilustración 4.20.

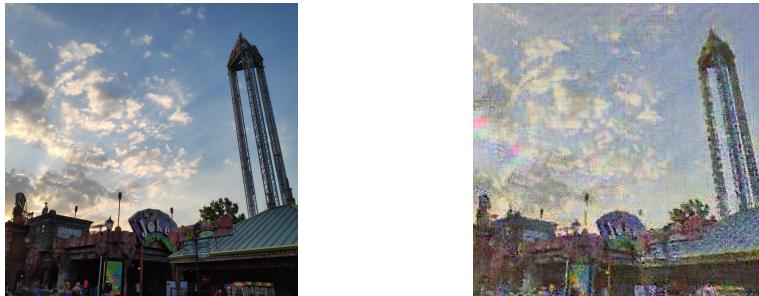


Figura 4.19: Cuadro de Monet generado en el parque Warner



Figura 4.20: Cuadro de Monet generado en el Palacio de Cristal de Madrid

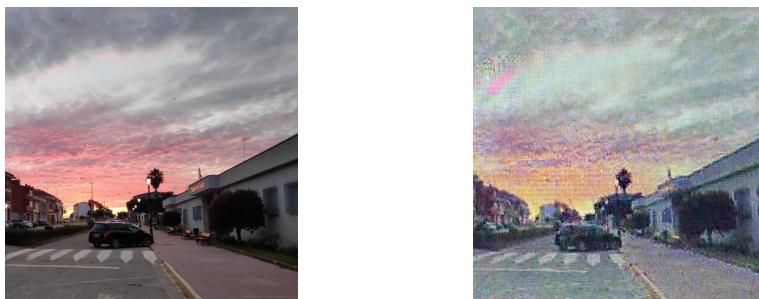


Figura 4.21: Cuadro de Monet generado en el centro de salud de Montehhermoso (Cáceres)

4.1.1.2.3 Van Gogh

Es especialmente notable el cambio de colores, dando paso a colores más chillones (figuras 4.23 y 4.24) y nuevamente las texturas (ilustración 4.22).



Figura 4.22: Cuadro de Van Gogh generado en la estación de Chamartín (Madrid)



Figura 4.23: Cuadro de Van Gogh generado a partir de una foto del autor en Hervás (Cáceres)

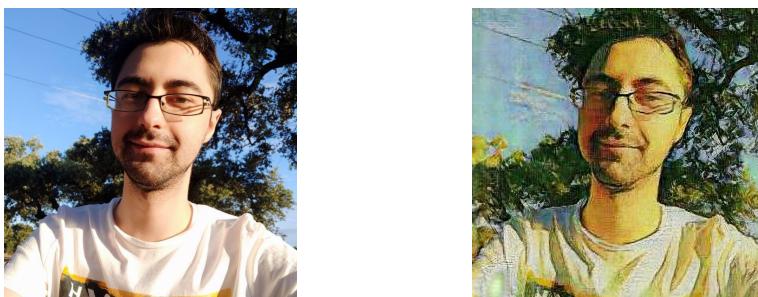


Figura 4.24: Cuadro de Van Gogh generado a partir de una foto del autor en la Dehesa Boyal de Montelhermoso (Cáceres)

4.1.2. Paso de cuadros a fotografías

Si bien esta transformación no es objetivo del presente Trabajo Fin de Grado, se incluye como reflexión acerca de la transformación adicional que nos brinda el modelo Cycle GAN.

4.1.2.1. Cézanne

Puede verse en las figura 4.26 y 4.27 que puede empezar a parecerse a una foto, ya que deshace relativamente bien la paleta de colores de Cézanne, pero sin embargo puede verse que los árboles son demasiado grandes y con errores

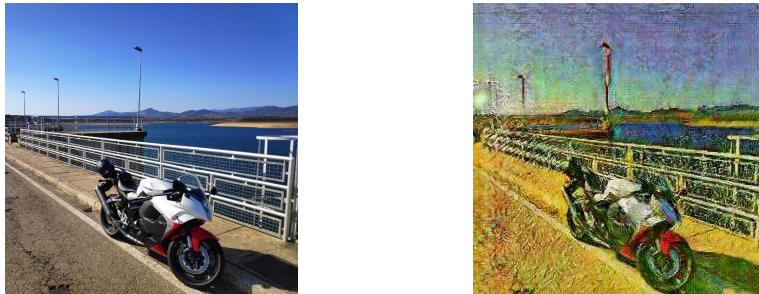


Figura 4.25: Cuadro de Van Gogh a partir de una Hyosung GTR 650 estacionada en el pantano de Gabriel y Galán (Cáceres)

gráficos, tapando las casas y generando por tanto una foto relativamente incorrecta (no olvidemos los planos superpuestos característicos de Cézanne, precursores del cubismo).

Por último puede verse en la figura 4.28 que el modelo es realmente malo a la hora de trabajar con retratos pictóricos, dejando al hombre como una mera sombra.



Figura 4.26: Foto basada en el cuadro de Cézanne *Marronniers et ferme au Jas de Bouffan*



Figura 4.27: Foto basada en el cuadro de Cézanne *Landscape. Study after Nature*

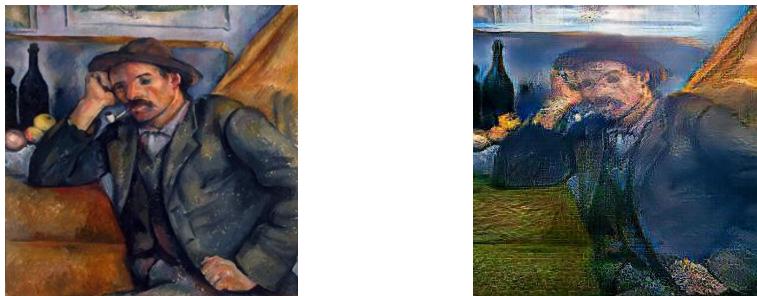


Figura 4.28: Foto basada en el cuadro de Cézanne *Homme à la pipe*

4.1.2.2. Monet

En el apartado anterior vimos que las transformaciones de Monet eran las menos agresivas, por lo que tiene sentido que las transformaciones inversas queden relativamente mejor que las de otros pintores. Podemos ver como aparece una paleta de colores con más contraste, especialmente en los tonos azules: mar de la figura 4.29 y cielo de las figuras 4.30 y 4.30.



Figura 4.29: Foto basada en el cuadro de Monet *Callejón cerca de Pourville*



Figura 4.30: Foto basada en el cuadro de Monet *The Promenade at Argenteuil*



Figura 4.31: Foto basada en el cuadro de Monet *El Verano*

4.1.2.3. Van Gogh

Si bien quizás los cuadros de Van Gogh eran los que generaba mejor el modelo, vemos su contrapartida en esta parte: la foto de los girasoles (figura 4.34) es bastante irrealista, con colores antinaturales, y con abundantes errores de texturas en las figuras 4.32 y 4.33.



Figura 4.32: Foto basada en el cuadro de Van Gogh *Un par de zuecos de cuero*



Figura 4.33: Foto basada en el cuadro de Van Gogh *Orchard Bordered by Cypress*



Figura 4.34: Foto basada en el cuadro de Van Gogh *Jarrón con catorce girasoles*

4.2. Objetivos logrados

El objetivo principal se ha podido ver que se ha cumplido, ya que el sistema funciona correctamente y se ha seguido una arquitectura con la que se pueden cambiar parámetros del modelo fácilmente. Artísticamente dista de emular fielmente el Impresionismo, pero podemos ver que con los avances en materia de redes de neuronas que se producen cada año quizás estemos más cerca de conseguirlo de lo que pueda parecer.

Pasemos a hablar de los objetivos secundarios (cada punto se refiere al número de objetivo planteado):

1. Dicha reflexión se ha realizado tanto en este apartado como en el capítulo conclusiones.
2. La implementación del modelo se ha realizado utilizando las herramientas más eficientes tanto de Python como de Keras y TensorFlow:
 - Python: uso del campo `_slots_` en la definición de las clases [58], ficheros `.json` de configuración del modelo, uso de la clase `Pathlib` para garantizar la compatibilidad entre sistemas operativos, patrones de diseño como `Singleton` para el ahorro de memoria, ficheros `requirements.txt` especificando las dependencias del proyecto para ser instaladas con `pip...`
 - Keras: uso de funciones para mostrar la arquitectura del modelo implementado y de la exportación de `checkpoints` para restaurar el entrenamiento del modelo.
 - TensorFlow: uso de su módulo `Dataset` para la optimización de la lectura y el procesado de imágenes [59].
3. Para cambiar el dataset únicamente hay que indicarlo en los parámetros de lanzadera y realizar como máximo dos cambios en el fichero `.json` de configuración.
4. Este objetivo se ha cumplido por completo al realizar los entrenos de la red utilizando instancias de cómputo y `buckets` para el almacenamiento de los `logs` en Google Cloud Platform.

5. Gracias al *microframework* Flask se ha cumplido este objetivo de forma muy sencilla e intuitiva: con una adaptación mínima podríamos desplegar el subsistema en un servidor externo y añadir nueva funcionalidad al mismo.
6. El cumplimiento de este último objetivo es trivial al realizarse este documento desde 0, sin ninguna plantilla.

4.3. Problemas encontrados

- Imposibilidad de utilizar los pesos de la red EnhanceNet en otra implementación: este factor ha imposibilitado hacer más eficiente la propia implementación, además de no poder convertirla a TensorFlow 2, lo que ha forzado a utilizar versiones depreciadas de algunas librerías.
- Entrenamiento de las redes CycleGAN muy costoso, lo que forzó a utilizar la instancia en Google Cloud Platform con la promoción de nuevo usuario: entre 14 y 40h, según versiones en la instancia de la nube, entre 40 y 80 en el equipo del autor. Este factor limitante ha restringido enormemente la experimentación con el modelo, además de condicionar partes del diseño final de la propia implementación.
- Consumo elevado de recursos de GPU por parte del modelo, lo que ha limitado experimentos con la resolución de las imágenes.
- Dificultad de implementar y depurar la red. Debido a la complejidad intrínseca al modelo y a las peculiaridades de TensorFlow me resultó muy complicado hacer funcionar el modelo y depurar los errores en la transformación de imágenes.
- Métricas poco representativas: si bien los baremos de la función error de CycleGAN son comprensibles, cuesta extrapolarlos a las imágenes que arroja el modelo, lo que dificulta aún más la mejora del mismo.
- Documentación escasa para instalar correctamente CUDA y cuDNN.

Capítulo 5

Conclusiones y trabajos futuros

5.1. Conclusiones

En primer lugar, podemos ver que la Inteligencia Artificial carece actualmente de la creatividad de los seres humanos, al no realizar transformaciones radicales de los objetos de las imágenes. En el capítulo anterior hemos podido ver que el modelo, especialmente en los cuadros que genera de Monet, básicamente realiza un filtro de color (ver figuras 4.7 y 4.8). Dicho filtro de colores consiste básicamente en las paletas de colores propias de los autores: en Monet es azulada, mientras que en Cézanne es rojiza y en Van Gogh amarillenta chillona.

También podemos ver que el modelo aprende de forma ciertamente irregular según el pintor: a diferencia de Monet, en el caso de Cézanne vemos como el sistema es capaz de modificar texturas (ver ilustraciones 4.1, 4.14 y 4.13) y formas de árboles (apreciable en las figuras 4.15 y 4.16). Por otra parte, en el caso de Van Gogh, es muy notable el aprendizaje del modelo de sus pinceladas bruscas: se puede ver con muchísima claridad en el cielo de la figura 4.22, en los árboles de la ilustración 4.23 y en el asfalto de la carretera de 4.25.

No obstante, no es todo oro lo que reluce. La implementación ha cometido un error (que el autor no ha sido capaz de detectar su causa y solucionarlo) de iluminación, lo que provoca que las partes más brillantes de las transformaciones tengan unas manchas blancas y/o rojizas, las cuales son bastante molestas.

Asimismo, puede verse que en las transformaciones de cuadro a foto hay gran cantidad de problemas gráficos, con texturas ciertamente antinaturales, salvo en Monet. En los cambios de cuadro a foto de este último vemos que ayuda su estilo de pintura al aire libre, con texturas suaves que el modelo es capaz de adaptar con relativa eficacia al dominio del mundo real. Esto provoca que tengamos resultados ciertamente creíbles, como puede ser la figura 4.30.

Podemos concluir que el modelo tiene cierto éxito para emular a los pintores, pero aún le queda mucho recorrido para resultar creíble para un ojo humano. De alguna manera el modelo se muestra empeñado en forzar la textura del óleo sobre lienzo, resultando en efectos visuales bastante extraños en ocasiones (ver parte izquierda de la figura 4.25).

No obstante, debido a los enormes avances de las redes GAN de los últimos años, cabe la posibilidad de que un nuevo modelo o mejorando el aquí implementado puedan corregirse los errores visuales.

5.2. Impacto social y medioambiental

A lo largo de este Trabajo Fin de Grado se ha podido ver que existe tecnología suficiente para emular con cierto éxito imágenes propias de determinados contextos, únicamente tendríamos que cambiar los datos de entrenamiento. Podríamos pensar que la eficacia de estos sistemas podría desbancar o perjudicar a los pintores tradicionales, ya que su trabajo puede perder valor al poder generarse automáticamente.

Este *temor* generalizado a que la Inteligencia Artificial y la Robótica nos quiten el empleo no es infundado, ya que medios importantes del país están hablando de este tema en sus columnas de opinión.

Los robots no son nuestros enemigos, sino una de las claves para incrementar nuestra productividad y nuestro bienestar.

Juan Ramón Rallo (economista), en su columna *¿Los robots nos están quitando el empleo?* [60].

La revolución de los robots generará desempleo, pero también riqueza, la clave está en el reparto.

Dario Pescador, en su artículo *Cómo te van a quitar tu trabajo los robots* [61].

La implantación generalizada de robots e inteligencias artificiales puede destruir empleos directos, pero a su vez generar empleos indirectos, como por ejemplo el mantenimiento y mejora del sistema planteado en este Trabajo Fin de Grado. En ese sentido, podemos argumentar que el sistema desarrollado puede imitar tendencias pictóricas a partir de fotografías, pero ambas fuentes tienen que existir previamente. El estilo del fotógrafo y el del pintor prevalecen, ya que a día de hoy el sistema no es capaz de desarrollar creatividad propiamente dicha.

El fusionar dos estilos diferenciados (fotógrafo y pintor) abre las puertas a nuevas formas de creación artística. Sin ir más lejos, jugando con los datos disponibles de este Trabajo Fin de Grado podríamos crear un *pintor metaimpresionista*, que posea los rasgos de Monet, Van Gogh y Cézanne. Podemos establecer que **la Inteligencia Artificial no es un reemplazo del artista, sino una herramienta que lo complementa**.

Por otra parte, en este Trabajo Fin de Grado se ha abordado la vertiente más *simpática* y artística de esta tecnología, pero las redes GAN, explicadas y utilizadas en este Trabajo Fin de Grado pueden utilizarse para fines más *oscuros*. El problema de los *deepfakes* (falsificaciones profundas en castellano) ya está aquí [62], existiendo vídeos de dirigentes políticos como Barack Obama pronunciando discursos falsos. Esto abre las puertas a nuevas formas de desinformación y manipulación política, con el enorme peligro que supone para nuestra democracia.

Si bien es cierto que hay aplicaciones benévolas, como la creación de muestras médicas para mejorar sistemas de reconocimiento, no podemos descuidar los avances en este sentido. Podcasts como XRey con sus experimentos creando discursos falsos de Franco (pronunciados por él mismo) revelan que es necesario concienciar a la ciudadanía de las ventajas y de los peligros de esta tecnología que supone para nuestra sociedad.

Por otra parte hemos podido ver que el sistema es relativamente costoso (en tiempo y en *hardware*) para un usuario promedio. Sin embargo dichos recursos son fácilmente obtenibles por organizaciones de cierta envergadura, como nuestra escuela, lo que plantea desigualdades de oportunidades para acceder a estos sistemas avanzados.

Asimismo, la necesidad de recurrir a herramientas propietarias como CUDA y de soluciones en la nube como Google Cloud Platform (especiamente para usuarios que no dispongan de gran capacidad de cómputo en sus ordenadores personales) agrava la dependencia tecnológica que tiene nuestra sociedad hacia las empresas, siendo en su enorme mayoría empresas fuera de la Unión Europea.

En este sentido se torna cada vez más necesaria la inversión a nivel europeo en la creación de *hardware* para reducir nuestra dependencia tecnológica del exterior, más importante aún en estos tiempos convulsos a nivel geopolítico. La compra de empresas como ARM por parte de nVidia [63] aviva esta necesidad, amén del impacto que tienen las grandes empresas tecnológicas estadounidenses en el desarrollo de tecnologías europeas a nivel de tratamiento de datos y de medios de cómputo. Vemos que los recursos *hardware* están disponibles en cada vez menos manos, por lo que urge recuperar la soberanía europea (tanto de datos como de computación) a través de diversas medidas.

Medioambientalmente hablando podemos esgrimir que la máquina puede consumir mucho tiempo de cómputo para desarrollar el entrenamiento del modelo y por tanto gran cantidad de electricidad. Sin embargo, en países como España no es necesariamente un impacto negativo. Podríamos desarrollar una solución en la cual se utilice energía solar, entre otras energías renovables, para generar electricidad que podría utilizar el computador mediante una batería.

Por otra parte, con la implantación masiva de soluciones basadas en computación distribuida, podemos realizar nuestras imágenes con un consumo prácticamente nulo en nuestro terminal (por ejemplo nuestro móvil), recayendo el coste de inferencia en el servidor. Dicho servidor podríamos situarlo en un *datacenter* alimentado por una planta de generación de electricidad renovable, lo que nos permite atajar otro gran problema: el problema demográfico que azota a nuestro país (comúnmente llamado *España vaciada*), al situar esas plantas de electricidad en localidades *vaciadas* de nuestro país, como Galisteo y Valdeobispo (provincia de Cáceres) [64].

5.3. Lineas futuras

Aunque este Trabajo Fin de Grado se ha basado en la implementación del paper [49] y en la construcción de un sistema software alrededor de él, eso no cierra las puertas a posibles ampliaciones y mejoras del proyecto, sino todo lo contrario. El autor propone las siguientes líneas futuras:

- Integrar de forma más natural el componente de ampliación de resolución: desarrollo íntegro del sistema en TensorFlow 2 y Keras.
- Construcción de un programa que, a través de una interfaz gráfica, proporcione una mejor experiencia de uso. Para ello se propone adaptar el sistema actual a una arquitectura cliente servidor.
- Mejorar y refinar la implementación para solucionar los problemas de iluminación.
- Utilizar este programa como concepto de estudio por parte de expertos en Impresionismo, con el objetivo de realizar una comparación artística entre el Impresionismo original y los resultados del presente sistema.
- Entrenar al sistema con resoluciones superiores para ofrecer mejores resultados. No obstante, por las limitaciones en las VRAM, sería necesario modificar la implementación para que seleccione automáticamente la máxima resolución con la que podría trabajar.
- Añadir nuevos pintores, ya sea pintores no expuestos en este Trabajo Fin de Grado o combinaciones de los ya mencionados. Podríamos estudiar cómo pintaría el sistema aprendiendo las técnicas de Monet, Van Gogh y Cézanne simultáneamente y compararlos con los resultados aquí obtenidos.

Bibliografía

- [1] Alan M. Turing. Lecture to the London Mathematical Society on 20 February 1947. Technical report, 1947.
- [2] Desconocido. Passport photo of Alan Turing at age 16, 1928.
- [3] Editorial Salvat. *Historia del Arte: El realismo. El impresionismo.* 2006.
- [4] Édouard Manet. Almuerzo sobre la hierba, 1863.
- [5] Cade Metz. Google's AI Wins Fifth And Final Game Against Go Genius Lee Sedol, 2016.
- [6] Pablo M. Diez. El «Gran Hermano» chino lo ve todo con cámaras que reconocen caras en segundos, 2019.
- [7] Belén García-Botija Aldana. Diagnóstico de la enfermedad de Parkinson usando deep learning y grabaciones de voz mediante teléfono móvil. *Escuela Técnica Superior de Ingenieros de Sistemas Informáticos (Universidad Politécnica de Madrid)*, page 52, 2019.
- [8] Toni Garcia. ¿Puede crear arte una máquina?, 2017.
- [9] Francisco Serradilla. ¿Pueden crear las máquinas? Quizás te sorprenda la respuesta, feb 2017.
- [10] Todocuadros. Impresionismo, pintura, historia y cuadros impresionistas.
- [11] Regina Sienra. ¿Qué es el impresionismo? Ejemplos y definición del impresionismo, 2019.
- [12] Claude Monet. Impresión, Sol Naciente, 1873.
- [13] Jean-François Millet. Las espigadoras, 1857.
- [14] Frans Hals. Grupo familiar ante un paisaje, 1648.
- [15] Paul Cézanne. El Sena en Bercy, 1878.
- [16] Claude Monet. La terraza de Sainte-Adresse, 1867.
- [17] Miguel Calvo Santos. Vincent Van Gogh, sep 2016.
- [18] Auguste Renoir. Retrato de Claude Monet, 1875.
- [19] Claude Monet. Mujer con sombrilla mirando a la izquierda, 1886.
- [20] Miguel Calvo Santos. Claude Monet, sep 2016.
- [21] Van Gogh. Autorretrato, 1889.
- [22] D Barreira. Las incógnitas sobre la muerte de Van Gogh: ¿se suicidó de un disparo o lo asesinaron?, jun 2019.
- [23] Vicent Van Gogh. La noche estrellada, 1889.

- [24] Miguel Calvo Santos. Paul Cézanne , sep 2016.
- [25] Paul Cezanne. Autorretrato con sombrero arrugado, 1894.
- [26] Elsa María Elizalde Ocampo. Paul Cézanne: “el padre de todos nosotros”, 2020.
- [27] EFE. El óleo 'Lot y sus hijas', de Rubens, subastado por 52,4 millones de euros, jul 2016.
- [28] Paul Cézanne. Los Jugadores de cartas, 1893.
- [29] Wikiquote. Paul Cézanne.
- [30] Francisco Serradilla García. Trasparencias de Introducción a la Inteligencia Artificial. 2012.
- [31] Javier Pastor. Qué es la inteligencia artificial, 2018.
- [32] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems.* O'Reilly, 2 edition, 2019.
- [33] Sabari Balaji. Overfitting. Overfitting overview, 2019.
- [34] Pedro Ponce Cruz. *Inteligencia Artificial con aplicaciones a la ingeniería.* 2010.
- [35] Francisco Serradilla García. Aprendizaje Automático Redes de Neuronas. 2016.
- [36] Ricardo Mendoza. Entendiendo las Redes Neuronales Artificiales, 2019.
- [37] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [38] Ayyüce Kizrak. Comparison of Activation Functions for Deep Neural Networks , may 2019.
- [39] Stuart Russell and Peter Norvig. *Inteligencia Artificial. Un Enfoque Moderno.* 2004.
- [40] Geoffrey Hilton, David Rumelhart, and Ronald Williams. Learning representations by back-propagating errors. *Nature Biotechnology*, 1986.
- [41] Nahua Kang. Multi-Layer Neural Networks with Sigmoid Function, jun 2017.
- [42] D H Hubel and T N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, 148(3):574–591, 1959.
- [43] Kunihiko Fukushima. Biological cybernetics neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, 1980.
- [44] Diego Calvo. Red Neuronal Convolucional, jul 2020.

- [45] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. In *Advances in Neural Information Processing Systems*, volume 3, pages 2672–2680, 2014.
- [46] Matías S. Zavia. ¿Cuál de estas personas es real y cuál fue generada por una IA?, dec 2018.
- [47] Francisco R. Villatoro. Caras humanas sintéticas hiperrealistas usando redes generativas antagónicas , dec 2018.
- [48] David Foster and an O'Reilly Media Company. Safari. *Generative deep learning : teaching machines to paint, write, compose, and play*. O'Reilly, 1 edition, 2019.
- [49] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. mar 2017.
- [50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9351, pages 234–241, 2015.
- [51] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778, 2016.
- [52] Marc Assens, Xavier Giro-I-Nieto, Kevin Mcguinness, and Noel E O'connor. PathGAN: Visual Scanpath Prediction with Generative Adversarial Networks. Technical report.
- [53] Neurohive. VGG16 - Convolutional Network for Classification and Detection, oct 2018.
- [54] Mehdi S. M. Sajjadi, Bernhard Schölkopf, and Michael Hirsch. EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis. dec 2016.
- [55] Shivani Rapole. EnhanceNet: Single Image Super-Resolution Through Automated Texture Synthesis, aug 2020.
- [56] Vince Tabora. Bicubic Interpolation Techniques For Digital Imaging, mar 2020.
- [57] Stack Overflow. Which TensorFlow and CUDA version combinations are compatible?, jul 2018.
- [58] Sébastien Chazallet. *Python 3: Los fundamentos del lenguaje*.
- [59] TensorFlow. tf.data.Dataset.
- [60] Juan Ramón Rallo. Empleo: ¿Los robots nos están quitando el empleo?, may 2019.

- [61] Darío Pescador. Cómo te van a quitar tu trabajo los robots, aug 2019.
- [62] IBM Developer Advocate in Silicon Valley. Deepfakes and the world of Generative Adversarial Networks, dec 2019.
- [63] Josep Roca. NVIDIA ha comprado ARM: ventajas y desventajas para el usuario, sep 2020.
- [64] El Periodico de la Energía. Cobra (ACS) promueve una planta fotovoltaica de 49,88 MW en Galisteo (Cáceres) , jul 2018.

Anexos

Apéndice A

Recursos hardware y software utilizados

A.1. Hardware

A.1.1. Equipo personal del autor

- CPU: Intel Core i7 7700HQ @ 3,80 GHz, con cuatro núcleos, litografía de 14 nm y 6 MB de caché.
- RAM: 8GB DDR4-2400
- HDD: 2.5”SATA con 1TB de almacenamiento a @7200 rpm
- SSD: Crucial MX500 M.2 con 500GB de almacenamiento.
- GPU: nVidia GTX 1050 con 2GB de VRAM GDDR5. Dispone de la arquitectura Pascal y de 640 núcleos CUDA.

A.1.2. Equipo contratado en Google Cloud Platform

- Tipo de máquina: n1-standard-4 (4 vCPU, 15 GB RAM)
- GPU: 1x nVidia Tesla T4, con 16GB GDDR6 de VRAM. Dispone de 2560 núcleos CUDA, 320 núcleos *Turing tensor* y de un ancho de banda de +320 GB/s
- SSD: 100GB
- Zona: us-west1-a

A.2. Software

- Sistemas Operativos: Ubuntu 18.04 LTS y Windows 10 versión 2004.
- Entorno de desarrollo de Python: PyCharm Professional Edition, versión 2020.2.3
- Python 3.6.9
- TensorFlow (y por extensión Tensorboard) 1.14 para el subsistema EnhanceNet y 2.2.1 para el sistema principal
- Keras 2.3.1
- Numpy 1.18.5

APÉNDICE A. RECURSOS HARDWARE Y SOFTWARE UTILIZADOS 68

- Flask 1.1.2
- Driver nVidia 450.51.05
- Cuda 11.0
- cuDNN 7.6.5
- Pillow 8.0.0

Apéndice B

Código fuente del proyecto

B.1. Sistema principal

El código de toda esta sección, además de en el presente anexo, está disponible en el GitHub personal del autor (VicDominguez), bajo el repositorio *TFG-Computadores-Principal*, accesible en la url <https://github.com/VicDominguez/TFG-Computadores-Principal>.

Ahí se pueden encontrar además los pesos de los modelos entrenados utilizados en este documento.

B.1.1. ampliar_imagen.py

```
import base64

import procesado_imagenes
import requests

def _ampliar_local(imagen, extension, factor_aumento):
    imagen_int = procesado_imagenes.float_array_a_int_array(imagen)
    return procesado_imagenes.numpy_array_a_imagen_PIL_bytes(
        procesado_imagenes.aumentar_resolucion(imagen_int,
                                                factor=factor_aumento), extension)

def ampliar(imagen, extension, factor_aumento, url_api=None):
    """Amplia una imagen. Según el valor de url_api,
    la amplia con la API de EnhanceNet o devuelve la imagen ampliada
    con el algoritmo de ampliación bicúbica.
    También se amplia con el algoritmo bicúbico si no es posible acceder a la
    API o se presenta alguna excepción en el intento.

Parámetros:
Imagen: imagen en formato array de numpy con valores entre 0 y 1.

Extensión: la extensión de la imagen anterior.

Factor aumento: La proporción para aumentar la resolución de la imagen.
Si se llama a la API de EnhanceNet,
este factor es ignorado.

Url_api: a qué url tenemos que realizar la petición de usarse la API.
Si es None, no se utiliza la API.
"""

if url_api is None:
    return _ampliar_local(imagen, extension, factor_aumento)
else:
    # preprocesamos la imagen
    imagen_formato_PIL = \
        procesado_imagenes.numpy_array_normalizado_a_imagen(imagen)
```

```

    imagen_base64 = \
        procesado_imagenes.imagen_a_base64_string(imagen_formato_PIL)
    imagen_base64 = str(imagen_base64, 'utf-8')
    # creamos la petición
    payload = {"imagen": imagen_base64}

    try:
        # intentamos redimensionar con la api
        # y si hay errores en el intento realizamos el método bicubico
        resultado_peticion = requests.post(url_api, json=payload)
        resultado_peticion = resultado_peticion.json()
        imagen_redimensionada_bytes = base64.b64decode(
            resultado_peticion["imagen_ampliada"])
    except:
        return _ampliar_local(imagen, extension, factor_aumento)

    return imagen_redimensionada_bytes

```

B.1.2. capas_extra.py

```

from keras.layers import InputSpec, Layer
import tensorflow as tf

class ReflectionPadding2D(Layer):
    """Clase que crea una capa para la arquitectura resnet. Fuente:
    https://github.com/davidADSP/GDL_code/blob/master/models/layers.py"""

    def __init__(self, padding=(1, 1), **kwargs):
        self.padding = tuple(padding)
        self.input_spec = [InputSpec(ndim=4)]
        super(ReflectionPadding2D, self).__init__(**kwargs)

    def compute_output_shape(self, s):
        """ If you are using "channels_last" configuration"""
        return s[0], s[1] + 2 * self.padding[0], \
               s[2] + 2 * self.padding[1], s[3]

    def call(self, y, mask=None):
        w_pad, h_pad = self.padding
        return tf.pad(y, [[0, 0], [h_pad, h_pad], [w_pad, w_pad],
                         [0, 0]], 'REFLECT')

```

B.1.3. cargador_imagenes.py

```

import tensorflow as tf

from utilidades import *
import procesado_imagenes

class CargadorImagenes(metaclass=Singleton):
    """Clase que sirve para alimentar de imágenes a la red cuando entrena"""

    __slots__ = ["tamanio_batch", "tamanio_buffer", "dimensiones",
                "logger", "numero_imagenes_entreno_pintor",
                "numero_imagenes_entreno_foto", "numero_imagenes_test_pintor",
                "numero_imagenes_test_foto", "dataset_entreno_pintor",
                "dataset_entreno_foto", "dataset_test_pintor",
                "dataset_test_foto"]

```

```

def __init__(self):
    utils = Utilidades()

    self.tamano_batch = utils.obtener_tamano_batch()
    self.tamano_buffer = utils.obtener_tamano_buffer()
    self.dimensiones = utils.obtener_dimensiones()

    rutas_imagenes_entreno_pintor = \
        utils.obtener_rutas_imagenes_entreno_pintor()
    rutas_imagenes_entreno_foto = \
        utils.obtener_rutas_imagenes_entreno_foto()
    rutas_imagenes_test_pintor = utils.obtener_rutas_imagenes_test_pintor()
    rutas_imagenes_test_foto = utils.obtener_rutas_imagenes_test_foto()

    # Atributos para calcular n_batches de forma eficiente
    self.numero_imagenes_entreno_pintor = len(rutas_imagenes_entreno_pintor)
    self.numero_imagenes_entreno_foto = len(rutas_imagenes_entreno_foto)
    self.numero_imagenes_test_pintor = len(rutas_imagenes_test_pintor)
    self.numero_imagenes_test_foto = len(rutas_imagenes_test_foto)

    self.logger = utils.obtener_logger("lector_imagenes")

    inicio = timestamp()
    listado_dataset_entreno_pintor = \
        tf.data.Dataset.list_files(rutas_imagenes_entreno_pintor)
    self.logger.info("listado_dataset_entreno_pintor " +
                     str(timestamp() - inicio))

    inicio = timestamp()
    listado_dataset_entreno_foto = \
        tf.data.Dataset.list_files(rutas_imagenes_entreno_foto)
    self.logger.info("listado_dataset_entreno_foto " +
                     str(timestamp() - inicio))

    inicio = timestamp()
    listado_dataset_test_pintor = \
        tf.data.Dataset.list_files(rutas_imagenes_test_pintor)
    self.logger.info("listado_dataset_test_pintor " +
                     str(timestamp() - inicio))

    inicio = timestamp()
    listado_dataset_test_foto = \
        tf.data.Dataset.list_files(rutas_imagenes_test_foto)
    self.logger.info("listado_dataset_test_foto " +
                     str(timestamp() - inicio))

AUTOTUNE = tf.data.experimental.AUTOTUNE

inicio = timestamp()
imagenes_entreno_pintor = listado_dataset_entreno_pintor.map(
    self._preprocesar_imagen_dataset, num_parallel_calls=AUTOTUNE)
imagenes_entreno_foto = listado_dataset_entreno_foto.map(
    self._preprocesar_imagen_dataset, num_parallel_calls=AUTOTUNE)
imagenes_test_pintor = listado_dataset_test_pintor.map(
    self._preprocesar_imagen_dataset, num_parallel_calls=AUTOTUNE)
imagenes_test_foto = listado_dataset_test_foto.map(
    self._preprocesar_imagen_dataset, num_parallel_calls=AUTOTUNE)
self.logger.info("Mapeado datasets " + str(timestamp() - inicio))

inicio = timestamp()
self.dataset_entreno_pintor = self.preparar_dataset(

```

```

        imagenes_entreno_pintor, cache=True,
        ruta_cache=utils.obtener_archivo_cache("dataset_entreno_pintor"))
self.dataset_entreno_foto = self.preparar_dataset(
    imagenes_entreno_foto, cache=True,
    ruta_cache=utils.obtener_archivo_cache("dataset_entreno_foto"))
self.dataset_test_pintor = self.preparar_dataset(
    imagenes_test_pintor, cache=True,
    ruta_cache=utils.obtener_archivo_cache("dataset_test_pintor"))
self.dataset_test_foto = self.preparar_dataset(
    imagenes_test_foto, cache=True,
    ruta_cache=utils.obtener_archivo_cache("dataset_test_foto"))
self.logger.info("Preparación datasets " + str(timestamp() - inicio))

def calcular_n_batches(self, entreno=True):
    """Calcula los n_batches del proceso a realizar.

    Parámetros:
        entreno: indicación si de el proceso es entrenamiento o no."""

    if entreno:
        imagenes_pintor = self.numero_imagenes_entreno_pintor
        imagenes_foto = self.numero_imagenes_entreno_foto
    else:
        imagenes_pintor = self.numero_imagenes_test_pintor
        imagenes_foto = self.numero_imagenes_test_foto

    return int(min(imagenes_pintor, imagenes_foto) / self.tamano_batch)

def cargar_batch(self, entreno=True):
    """Generador que devuelve imágenes en formato numpy
    para realizar un proceso batch.

    Parámetros:
        entreno: indicación de si el proceso es entrenamiento o no."""

    if entreno:
        dataset_pintor = self.dataset_entreno_pintor
        dataset_foto = self.dataset_entreno_foto
    else:
        dataset_pintor = self.dataset_test_pintor
        dataset_foto = self.dataset_test_foto

    n_batches = self.calcular_n_batches(entreno)

    iter_dataset_pintor = iter(dataset_pintor)
    iter_dataset_foto = iter(dataset_foto)

    for i in range(n_batches):
        yield next(iter_dataset_pintor).numpy(), \
            next(iter_dataset_foto).numpy()

def preparar_dataset(self, dataset, cache, ruta_cache=None):
    """Prepara el dataset para ser consumido.

    Parámetros:
        dataset: el dataset a preparar (tf.Dataset)
        cache: si queremos usar o no caché.
        ruta_cache: si hemos indicado que queremos usar caché,
                    parámetro indica la ruta a guardar
                    la caché si no encaja en memoria."""


```

```

AUTOTUNE = tf.data.experimental.AUTOTUNE
if cache:
    # verdadero si el parámetro es un string, un número distinto de 0..
    if isinstance(ruta_cache, str):
        dataset = dataset.cache(ruta_cache)
    else:
        dataset = dataset.cache()
dataset = dataset.shuffle(buffer_size=self.tamanio_buffer)
dataset = dataset.repeat() # repetimos continuamente
dataset = dataset.batch(self.tamanio_batch)
dataset = dataset.prefetch(buffer_size=AUTOTUNE)
return dataset

def _preprocesar_imagen_dataset(self, ruta):
    """Recubridor para llamar a la función procesar
    imagen desde la función map.

    Parámetros:
        ruta: ruta en disco de la imagen"""
    return procesado_imagenes.preprocesar_imagen(ruta, self.dimensiones)

```

B.1.4. cyclegan.py

```

from __future__ import division, print_function

import pickle as pkl

import matplotlib.pyplot as plt
import tensorflow as tf
from keras.initializers import RandomNormal
from keras.layers import Activation, Concatenate, Dropout, Input
from keras.layers.advanced_activations import LeakyReLU
from keras.layers.convolutional import Conv2D, Conv2DTranspose, UpSampling2D
from keras.layers.merge import add
from keras.models import Model, load_model
from keras.optimizers import Adam
from keras.utils import plot_model
from keras_contrib.layers.normalization.instancenormalization \
    import InstanceNormalization
import numpy as np
import io

from ampliar_imagen import ampliar
from procesado_imagenes import preprocesar_imagen_individual, normalizar_float
from utilidades import *
from capas_extra import ReflectionPadding2D

class CycleGAN:
    __slots__ = ['utils', 'logger', "tipo_generador", "tasa_aprendizaje",
                "lambda_validacion", "lambda_reconstruccion",
                "lambda_identidad", "filtros_generador", "filtros_discriminador",
                "dimensiones", "epoch_actual", "disc_patch",
                "inicializacion_pesos", "discriminador_pintor",
                "discriminador_foto", "generador_foto",
                "generador_pintor", "modelo_combinado"]

    def __init__(self, restaurar=False, tipo_generador="resnet"):

        def crear_generador_unet():

            def downsample(capa_entrada, filtros, tamanio_filtro=4):

```

```

d = Conv2D(filtros, kernel_size=tamano_filtro, strides=2,
           padding='same')(capa_entrada)
d = InstanceNormalization(axis=-1, center=False,
                           scale=False)(d)
d = Activation('relu')(d)

return d

def upsample(capa_entrada, saltar_entrada, filtros,
             tamano_filtro=4, ratio_dropout=0):
    u = UpSampling2D(size=2)(capa_entrada)
    u = Conv2D(filtros, kernel_size=tamano_filtro,
               strides=1, padding='same')(u)
    u = InstanceNormalization(axis=-1, center=False, scale=False)(u)
    u = Activation('relu')(u)
    if ratio_dropout:
        u = Dropout(ratio_dropout)(u)

    u = Concatenate()([u, saltar_entrada])
    return u

# Image input
img = Input(shape=self.dimensiones)

# Downsampling
d1 = downsample(img, self.filtros_generador)
d2 = downsample(d1, self.filtros_generador * 2)
d3 = downsample(d2, self.filtros_generador * 4)
d4 = downsample(d3, self.filtros_generador * 8)

# Upsampling
u1 = upsample(d4, d3, self.filtros_generador * 4)
u2 = upsample(u1, d2, self.filtros_generador * 2)
u3 = upsample(u2, d1, self.filtros_generador)

u4 = UpSampling2D(size=2)(u3)
salida_img = Conv2D(self.dimensiones[2], kernel_size=4, strides=1,
                     padding='same', activation='tanh')(u4)

return Model(img, salida_img)

def crear_generador_resnet():

    def conv7s1(capa_entrada, filtros, final):
        y = ReflectionPadding2D(padding=(3, 3))(capa_entrada)
        y = Conv2D(filtros, kernel_size=(7, 7), strides=1,
                   padding='valid',
                   kernel_initializer=self.inicializacion_pesos)(y)
        if final:
            y = Activation('tanh')(y)
        else:
            y = InstanceNormalization(axis=-1, center=False,
                                      scale=False)(y)
            y = Activation('relu')(y)
        return y

    def downsample(capa_entrada, filtros):
        y = Conv2D(filtros, kernel_size=(3, 3), strides=2,
                   padding='same',
                   kernel_initializer=self.inicializacion_pesos)(
            capa_entrada)
        y = InstanceNormalization(axis=-1, center=False,

```

```

        scale=False)(y)
y = Activation('relu')(y)
return y

def residual(capa_entrada, filtros):
    atajo = capa_entrada
    y = ReflectionPadding2D(padding=(1, 1))(capa_entrada)
    y = Conv2D(filtros, kernel_size=(3, 3), strides=1,
               padding='valid',
               kernel_initializer=self.inicializacion_pesos)(y)
    y = InstanceNormalization(axis=-1, center=False,
                               scale=False)(y)
    y = Activation('relu')(y)

    y = ReflectionPadding2D(padding=(1, 1))(y)
    y = Conv2D(filtros, kernel_size=(3, 3), strides=1,
               padding='valid',
               kernel_initializer=self.inicializacion_pesos)(y)
    y = InstanceNormalization(axis=-1, center=False,
                               scale=False)(y)

    y = Activation('relu')(y)

    return add([atajo, y])

def upsample(capa_entrada, filtros):
    y = Conv2DTranspose(filtros, kernel_size=(3, 3),
                        strides=2, padding='same',
                        kernel_initializer=self.inicializacion_pesos)(
        capa_entrada)
    y = InstanceNormalization(axis=-1, center=False, scale=False)(y)
    y = Activation('relu')(y)

    return y

# Image input
img = Input(shape=self.dimensiones)

x = img

x = conv7s1(x, self.filtros_generador, False)
x = downsample(x, self.filtros_generador * 2)
x = downsample(x, self.filtros_generador * 4)
x = residual(x, self.filtros_generador * 4)
x = upsample(x, self.filtros_generador * 2)
x = upsample(x, self.filtros_generador)
x = conv7s1(x, 3, True)
salida = x

return Model(img, salida)

def crear_discriminador():

    def conv4(capa_entrada, filtros, stride=2, norm=True):
        y = Conv2D(filtros, kernel_size=(4, 4), strides=stride,
                   padding='same',

```

```

        kernel_initializer=self.inicializacion_pesos)(
    capa_entrada)

    if norm:
        y = InstanceNormalization(axis=-1, center=False,
                                  scale=False)(y)

    y = LeakyReLU(0.2)(y)

    return y

    img = Input(shape=self.dimensiones)

    x = conv4(img, self.filtros_discriminador, stride=2, norm=False)
    x = conv4(x, self.filtros_discriminador * 2, stride=2)
    x = conv4(x, self.filtros_discriminador * 4, stride=2)
    x = conv4(x, self.filtros_discriminador * 8, stride=1)

    salida = Conv2D(1, kernel_size=4, strides=1, padding='same',
                    kernel_initializer=self.inicializacion_pesos)(x)

    return Model(img, salida)

self.utils = Utilidades()
self.logger = self.utils.obtener_logger("cyclegan")
self.dimensiones = self.utils.obtener_dimensiones()

if restaurar and self.utils.existen_ficheros_modelo():
    self.logger.info("Cargando modelo")
    self.discriminador_pintor = load_model(
        self.utils.obtener_ruta_fichero_discriminador_pintor(),
        custom_objects={'InstanceNormalization': InstanceNormalization})
    self.discriminador_foto = load_model(
        self.utils.obtener_ruta_fichero_discriminador_foto(),
        custom_objects={'InstanceNormalization': InstanceNormalization})
    self.generador_pintor = load_model(
        self.utils.obtener_ruta_fichero_generador_pintor(),
        custom_objects={'ReflectionPadding2D': ReflectionPadding2D,
                       'InstanceNormalization': InstanceNormalization})
    self.generador_foto = load_model(
        self.utils.obtener_ruta_fichero_generador_foto(),
        custom_objects={'ReflectionPadding2D': ReflectionPadding2D,
                       'InstanceNormalization': InstanceNormalization})
    self.modelo_combinado = load_model(
        self.utils.obtener_ruta_fichero_modelo(),
        custom_objects={'ReflectionPadding2D': ReflectionPadding2D,
                       'InstanceNormalization': InstanceNormalization})
    self.logger.info("Modelo cargado")
else:
    self.logger.info("Creando modelo")
    self.tipo_generador = tipo_generador

    # obtenemos parámetros del modelo
    self.tasa_aprendizaje = self.utils.obtener_tasa_aprendizaje()
    self.lambda_validacion = self.utils.obtener_lambda_validacion()
    self.lambda_reconstruccion = self.utils.obtener_lambda_reconstruccion()
    self.lambda_identidad = self.utils.obtener_lambda_identidad()
    self.filtros_generador = self.utils.obtener_filtros_generador()
    self.filtros_discriminador = self.utils.obtener_filtros_discriminador()
    self.epoch_actual = 1

    # Calculate salida shape of D (PatchGAN)

```

```

patch = int(self.dimensiones[1] / 2 ** 3)
self.disc_patch = (patch, patch, 1)

self.inicializacion_pesos = RandomNormal(mean=0., stddev=0.02)

# Creamos los discriminadores
self.discriminador_pintor = crear_discriminador()
self.discriminador_foto = crear_discriminador()

self.discriminador_pintor.compile(loss='mse',
                                    optimizer=Adam(
                                        self.tasa_aprendizaje, 0.5),
                                    metrics=['accuracy'])
self.discriminador_foto.compile(loss='mse',
                                    optimizer=Adam(
                                        self.tasa_aprendizaje, 0.5),
                                    metrics=['accuracy'])

# Creamos los generadores
if self.tipo_generador == 'unet':
    self.generador_foto = crear_generador_unet()
    self.generador_pintor = crear_generador_unet()
else:
    self.generador_foto = crear_generador_resnet()
    self.generador_pintor = crear_generador_resnet()

# Para el modelo combinado solo entrenamos los generadores
self.discriminador_pintor.trainable = False
self.discriminador_foto.trainable = False

self.modelo_combinado = self._crear_modelo()

self.modelo_combinado.compile(loss=['mse', 'mse',
                                     'mae', 'mae',
                                     'mae', 'mae'],
                               loss_weights=[self.lambda_validacion,
                                             self.lambda_validacion,
                                             self.lambda_reconstruccion,
                                             self.lambda_reconstruccion,
                                             self.lambda_identidad,
                                             self.lambda_identidad],
                               optimizer=Adam(self.tasa_aprendizaje, 0.5))

self.discriminador_pintor.trainable = True
self.discriminador_foto.trainable = True
# cargamos los ultimos pesos, si hay
self.cargar_ultimos_pesos()

def train(self, lector_imagenes):

    sumario_entreno = tf.summary.create_file_writer(
        self.utils.obtener_ruta_logs_entreno())
    sumario_test = tf.summary.create_file_writer(
        self.utils.obtener_ruta_logs_test())

    imagen_muestra_pintor = preprocesar_imagen_individual(
        self.utils.obtener_archivo_muestra_pintor(),
        self.dimensiones).numpy()
    imagen_muestra_foto = preprocesar_imagen_individual(
        self.utils.obtener_archivo_muestra_foto(),
        self.dimensiones).numpy()

```

```

# Umbrales para los errores adversativos
umbral_verdad = np.ones((1,) + self.disc_patch)
umbral_falso = np.zeros((1,) + self.disc_patch)

numero_batches_entreno = lector_imagenes.calcular_n_batches(entreno=True)
numero_batches_test = lector_imagenes.calcular_n_batches(entreno=False)

# Inicializamos los arrays que contendrán los errores
error_discriminadores_entreno = np.ones(numero_batches_entreno)
precision_discriminadores_entreno = np.ones(numero_batches_entreno)
error_generadores_entreno = np.ones(numero_batches_entreno)
validez_generadores_entreno = np.ones(numero_batches_entreno)
error_reconstruccion_generadores_entreno = \
    np.ones(numero_batches_entreno)
error_identidad_generadores_entreno = np.ones(numero_batches_entreno)

error_discriminadores_test = np.ones(numero_batches_test)
precision_discriminadores_test = np.ones(numero_batches_test)
error_generadores_test = np.ones(numero_batches_test)
validez_generadores_test = np.ones(numero_batches_test)
error_reconstruccion_generadores_test = np.ones(numero_batches_test)
error_identidad_generadores_test = np.ones(numero_batches_test)

hay_gsutil = gsutil_disponible()
# Booleano para saber si tenemos que escribir o no en gcp

# Comenzamos
comienzo_entrenamiento = timestamp()

for epoch in range(self.epoch_actual, self.utils.obtener_epochs() + 1):

    if hay_gsutil:
        self.utils.copiar_logs_gcp()

    comienzo_epoch = timestamp()
    self.logger.info("epoch " + str(epoch))

    # paso de entreno
    for indice, (imagenes_pintor, imagenes_foto) \
        in enumerate(lector_imagenes.cargar_batch(entreno=True)):
        error_discriminadores_actual = \
            self._entrenar_discriminadores(imagenes_pintor,
                                            imagenes_foto,
                                            umbral_verdad, umbral_falso)
        error_generadores_actual = self._entrenar_generadores(
            imagenes_pintor, imagenes_foto, umbral_verdad)

        error_discriminadores_entreno[indice] = \
            error_discriminadores_actual[0]
        precision_discriminadores_entreno[indice] = \
            100 * error_discriminadores_actual[1]
        error_generadores_entreno[indice] = error_generadores_actual[0]
        validez_generadores_entreno[indice] = \
            np.mean(error_generadores_actual[1:3])
        error_reconstruccion_generadores_entreno[indice] = \
            np.mean(error_generadores_actual[3:5])
        error_identidad_generadores_entreno[indice] = \
            np.mean(error_generadores_actual[5:6])

    # paso test
    for indice, (imagenes_pintor, imagenes_foto) in \
        enumerate(lector_imagenes.cargar_batch(entreno=False)):

```

```

        error_discriminadores_actual = \
            self._entrenar_discriminadores(imagenes_pintor,
                                            imagenes_foto,
                                            umbral_verdad, umbral_falso)
        error_generadores_actual = self._entrenar_generadores(
            imagenes_pintor, imagenes_foto, umbral_verdad)

        error_discriminadores_test[indice] = \
            error_discriminadores_actual[0]
        precision_discriminadores_test[indice] = \
            100 * error_discriminadores_actual[1]
        error_generadores_test[indice] = error_generadores_actual[0]
        validez_generadores_test[indice] = \
            np.mean(error_generadores_actual[1:3])
        error_reconstruccion_generadores_test[indice] = \
            np.mean(error_generadores_actual[3:5])
        error_identidad_generadores_test[indice] = \
            np.mean(error_generadores_actual[5:6])

    self._escribir_metricas_escalares(error_discriminadores_entreno,
                                      precision_discriminadores_entreno,
                                      error_generadores_entreno,
                                      validez_generadores_entreno,
                                      error_reconstruccion_generadores_entreno,
                                      error_identidad_generadores_entreno,
                                      sumario_entreno, epoch)

    self._escribir_metricas_escalares(error_discriminadores_test,
                                      precision_discriminadores_test,
                                      error_generadores_test,
                                      validez_generadores_test,
                                      error_reconstruccion_generadores_test,
                                      error_identidad_generadores_test,
                                      sumario_test, epoch)

    self._imagen_muestra(imagen_muestra_pintor, imagen_muestra_foto,
                         epoch)
fin_epoch = timestamp()

self.logger.info("epoch " + str(epoch) +
                 " completado en " + str(fin_epoch - comienzo_epoch))

if epoch % 5 == 0:
    self.logger.info("Guardando progreso")
    self._guardar_progreso(epoch)

self.epoch_actual += 1

fin_entrenamiento = timestamp()
self.logger.info("Entrenamiento completado en " +
                 str(fin_entrenamiento - comienzo_entrenamiento))
self._guardar_modelo()

def convertir_imagen(self, ruta_imagen, modo_destino, factor_aumento):
    # obtenemos la extensión de la imagen
    extension = ruta_imagen.split(".")[-1]
    if extension.lower() == "jpg":
        extension = "jpeg"
    # preprocesamos
    imagen = preprocessar_imagen_individual(ruta_imagen,
                                              self.dimensiones).numpy()
    # obtenemos la predicción

```

```

    imagen_predecida = self._predecir_imagen(imagen, modo_destino)
    # ampliamos y devolvemos
    return ampliar(imagen_predecida, extension, factor_aumento,
                   self.utils.obtener_url_api_aumento())

def _crear_modelo(self):
    # Entrada de los dos dominios
    imagen_pintor = Input(shape=self.dimensiones)
    imagen_foto = Input(shape=self.dimensiones)

    # Traducción al otro dominio
    falsificacion_foto = self.generador_foto(imagen_pintor)
    falsificacion_pintor = self.generador_pintor(imagen_foto)
    # Reconstrucción de ambos dominios
    reconstruccion_pintor = self.generador_pintor(falsificacion_foto)
    reconstruccion_foto = self.generador_foto(falsificacion_pintor)
    # Generación al mismo dominio
    imagen_pintor_identidad = self.generador_pintor(imagen_pintor)
    imagen_foto_identidad = self.generador_foto(imagen_foto)

    # Los discriminadores determinan la validez de las imágenes "traducidas"
    validador_pintor = self.discriminador_pintor(falsificacion_pintor)
    validador_foto = self.discriminador_foto(falsificacion_foto)

    # El modelo combinado entrena los generadores
    # para engañar a los discriminadores
    return Model(inputs=[imagen_pintor, imagen_foto],
                  outputs=[validador_pintor, validador_foto,
                           reconstruccion_pintor, reconstruccion_foto,
                           imagen_pintor_identidad, imagen_foto_identidad])

def _entrenar_discriminadores(self, imagenes_pintor, imagenes_foto,
                               umbral_verdad, umbral_falso):
    # Traducir imágenes
    falsificacion_foto = self.generador_foto.predict(imagenes_pintor)
    falsificacion_pintor = self.generador_pintor.predict(imagenes_foto)

    # TEntrenamos los discriminadores
    error_discriminador_pintor_real = \
        self.discriminador_pintor.train_on_batch(imagenes_pintor, umbral_verdad)
    error_discriminador_pintor_falsificacion = \
        self.discriminador_pintor.train_on_batch(
            falsificacion_pintor, umbral_falso)
    error_discriminador_foto_real = \
        self.discriminador_foto.train_on_batch(imagenes_foto, umbral_verdad)
    error_discriminador_foto_falsificacion = \
        self.discriminador_foto.train_on_batch(falsificacion_foto,
                                               umbral_falso)

    # Metricas
    error_discriminador_pintor = 0.5 * np.add(
        error_discriminador_pintor_real,
        error_discriminador_pintor_falsificacion)
    error_discriminador_foto = 0.5 * np.add(
        error_discriminador_foto_real,
        error_discriminador_foto_falsificacion)

    # Error total discriminador
    return 0.5 * np.add(error_discriminador_pintor, error_discriminador_foto)

def _test_discriminadores(self, imagenes_pintor,

```

```

    imagenes_foto, umbral_verdad, umbral_falso):

    # Traducir imagenes
    falsificacion_foto = self.generador_foto.predict(imagenes_pintor)
    falsificacion_pintor = self.generador_pintor.predict(imagenes_foto)

    # TEntrenamos los discriminadores
    error_discriminador_pintor_real = \
        self.discriminador_pintor.test_on_batch(imagenes_pintor,
                                                umbral_verdad)
    error_discriminador_pintor_falsificacion = \
        self.discriminador_pintor.test_on_batch(
            falsificacion_pintor, umbral_falso)
    error_discriminador_foto_real = \
        self.discriminador_foto.test_on_batch(imagenes_foto, umbral_verdad)
    error_discriminador_foto_falsificacion = \
        self.discriminador_foto.test_on_batch(falsificacion_foto,
                                              umbral_falso)

    # Metricas
    error_discriminador_pintor = 0.5 * np.add(
        error_discriminador_pintor_real,
        error_discriminador_pintor_falsificacion)
    error_discriminador_foto = 0.5 * np.add(
        error_discriminador_foto_real,
        error_discriminador_foto_falsificacion)

    # Error total discriminador
    return 0.5 * np.add(error_discriminador_pintor,
                         error_discriminador_foto)

def _entrenar_generadores(self, imagenes_pintor,
                           imagenes_foto, umbral_verdad):
    return self.modelo_combinado.train_on_batch(
        [imagenes_pintor, imagenes_foto],
        [umbral_verdad, umbral_verdad,
         imagenes_pintor, imagenes_foto,
         imagenes_pintor, imagenes_foto])

def _test_generadores(self, imagenes_pintor, imagenes_foto, umbral_verdad):
    return self.modelo_combinado.test_on_batch(
        [imagenes_pintor, imagenes_foto],
        [umbral_verdad, umbral_verdad,
         imagenes_pintor, imagenes_foto,
         imagenes_pintor, imagenes_foto])

def _predecir_imagen(self, imagen, modo_destino):
    if modo_destino.lower() == "pintor":
        imagen_predecida = self.generador_pintor.predict(imagen)
    else:
        imagen_predecida = self.generador_foto.predict(imagen)
    return normalizar_float(imagen_predecida)

def _imagen_muestra(self, imagen_pintor, imagen_foto, epoch):
    file_writer = tf.summary.create_file_writer(
        self.utils.obtener_ruta_logs())

    filas, columnas = 2, 3

    estimacion_foto = self._predecir_imagen(imagen_pintor, "foto")
    estimacion_pintor = self._predecir_imagen(imagen_foto, "pintor")
    pintor_reconstruido = self._predecir_imagen(estimacion_foto, "pintor")
    foto_reconstruida = self._predecir_imagen(estimacion_pintor, "foto")

```

```

    imagenes = np.concatenate(
        [imagen_pintor, estimacion_foto, pintor_reconstruido,
         imagen_foto, estimacion_pintor, foto_reconstruida])

    # Rescale images 0 - 1
    # imagenes = 0.5 * imagenes + 0.5

    titles = ['Original', 'Traducida', 'Reconstruida']
    figura, ejes = plt.subplots(filas, columnas)
    contador = 0
    for fila in range(filas):
        for columna in range(columnas):
            ejes[fila, columna].imshow(imagenes[contador])
            ejes[fila, columna].set_title(titles[columna], wrap=True)
            ejes[fila, columna].axis('off')
        contador += 1

    """Convierte un gráfico de matplotlib a un tensor
    de una imagen png para Tensorboard"""
    # Guardamos el grafico
    buffer = io.BytesIO()
    plt.savefig(buffer, format='png', dpi=400)
    # Cerramos la figura
    plt.close(figura)
    buffer.seek(0)
    # Pasamos el buffer png a un tensor primero 3D y
    # luego 4D, lo que necesita tensorflow
    imagen_tf = tf.image.decode_png(buffer.getvalue(), channels=4)
    imagen_tf = tf.expand_dims(imagen_tf, 0)

    with file_writer.as_default():
        tf.summary.image("Imagen resumen", imagen_tf, step=epoch)

def serializar_red(self):

    with open(self.utils.obtener_ruta_archivo_parametros(), 'wb') \
        as archivo:
        pkl.dump([self.tasa_aprendizaje, self.lambda_reconstruccion,
                  self.lambda_validacion, self.lambda_identidad,
                  self.dimensiones, self.filtros_generador,
                  self.filtros_discriminador],
                  archivo)

    plot_model(self.modelo_combinado,
               to_file=self.utils.obtener_ruta_archivo_modelo_esquema(),
               show_shapes=True, show_layer_names=True, dpi=300)
    plot_model(self.discriminador_pintor,
               to_file=self.utils.
               obtener_ruta_archivo_discriminador_pintor_esquema(),
               show_shapes=True, show_layer_names=True, dpi=300)
    plot_model(self.discriminador_foto, to_file=self.utils.
               obtener_ruta_archivo_discriminador_foto_esquema(),
               show_shapes=True, show_layer_names=True, dpi=300)
    plot_model(self.generador_pintor, to_file=self.utils.
               obtener_ruta_archivo_generador_pintor_esquema(),
               show_shapes=True, show_layer_names=True, dpi=300)
    plot_model(self.generador_foto, to_file=self.utils.
               obtener_ruta_archivo_generador_foto_esquema(),
               show_shapes=True, show_layer_names=True, dpi=300)

def _guardar_modelo(self):

```

```

        self.modelo_combinado.save(self.utils.obtener_ruta_fichero_modelo())
        self.discriminador_pintor.save(
            self.utils.obtener_ruta_fichero_discriminador_pintor())
        self.discriminador_foto.save(
            self.utils.obtener_ruta_fichero_discriminador_foto())
        self.generador_pintor.save(
            self.utils.obtener_ruta_fichero_generador_pintor())
        self.generador_foto.save(
            self.utils.obtener_ruta_fichero_generador_foto())

    with open(self.utils.obtener_ruta_archivo_modelo_objeto(), "wb") \
        as archivo:
        pkl.dump(self, archivo)

    def cargar_ultimos_pesos(self):
        self.logger.info("Cargamos los pesos más recientes")
        ruta_ultimo_checkpoint, self.epoch_actual = \
            self.utils.obtener_ultimos_pesos()
        self.logger.info("Los ultimos pesos eran del epoch: "
                        + str(self.epoch_actual))
        if ruta_ultimo_checkpoint is not None:
            self.modelo_combinado.load_weights(ruta_ultimo_checkpoint)

    def _guardar_progreso(self, epoch):
        self.modelo_combinado.save_weights(self.utils.
            obtener_ruta_fichero_modelo_por_epoch(epoch))

    @staticmethod
    def _escribir_metricas_escalares(error_discriminadores, precision_discriminadores,
                                      error_generadores, validez_generadores,
                                      error_reconstruccion_generadores,
                                      error_identidad_generadores, writer, step):

        with writer.as_default():
            tf.summary.scalar("Media del error de los discriminadores",
                               np.mean(error_discriminadores), step=step)
            tf.summary.scalar("Media de la precision de los discriminadores",
                               np.mean(precision_discriminadores),
                               step=step)
            tf.summary.scalar("Media del error de los generadores",
                               np.mean(error_generadores), step=step)
            tf.summary.scalar("Media del error de la validez de los generadores",
                               np.mean(validez_generadores),
                               step=step)
            tf.summary.scalar("Media del error de reconstruccion "
                              "de los generadores",
                              np.mean(error_reconstruccion_generadores),
                              step=step)
            tf.summary.scalar("Media del error de identidad "
                              "de los generadores",
                              np.mean(error_identidad_generadores),
                              step=step)
            tf.summary.scalar("Desviación estándar del error "
                              "de los discriminadores",
                              np.std(error_discriminadores),
                              step=step)
            tf.summary.scalar("Desviación estándar de la "
                              "precision de los discriminadores",
                              np.std(precision_discriminadores), step=step)
            tf.summary.scalar("Desviación estándar del error de "
                              "los generadores",
                              np.std(error_generadores), step=step)

```

```

    tf.summary.scalar("Desviación estándar de la validez de "
                      "los generadores",
                      np.std(validez_generadores), step=step)
    tf.summary.scalar("Desviación estándar del error de "
                      "reconstrucción de los generadores",
                      np.std(error_reconstrucción_generadores),
                      step=step)
    tf.summary.scalar("Desviación estándar del error "
                      "de identidad de los generadores",
                      np.std(error_identidad_generadores),
                      step=step)

```

B.1.5. lanzadera_entreno.py

```

import argparse
import pathlib

from cyclegan import CycleGAN
from cargador_imagenes import CargadorImagenes
from utilidades import Utilidades

if __name__ == "__main__":
    # Leemos los argumentos
    parser = argparse.ArgumentParser()
    parser.add_argument("-v", "--version",
                        help="Especifica el nombre de la versión "
                             "con la que se guardarán los archivos",
                        type=str,
                        default="version_por_defecto")
    parser.add_argument("-d", "--dataset",
                        help="Especifica el dataset a utilizar",
                        type=str,
                        default="monet2photo")
    parser.add_argument("-c", "--configuracion",
                        help="Especifica el fichero "
                             "de configuración que se va a usar",
                        type=str,
                        default="configuracion_128.json")
    parser.add_argument("-a", "--arquitectura",
                        help="Especifica el tipo de arquitectura: unet o resnet",
                        type=str,
                        default="resnet")
    args = vars(parser.parse_args())

    # Comprobamos que los parámetros son válidos
    assert pathlib.Path("../configuracion", args["configuracion"]).exists(), \
        "El archivo de configuración no existe"
    assert args["arquitectura"] in ["resnet", "unet"], "La arquitectura no es valida"

    utils = Utilidades(args["version"], args["dataset"], args["configuracion"])

    logger = utils.obtener_logger("lanzadera entrenamiento")

    utils.asegurar_dataset()
    logger.info("Dataset en linea")

    logger.info("Creacion de la red neuronal")
    gan = CycleGAN(restaurar=False, tipo_generador=args["arquitectura"])
    logger.info("Red neuronal lista")

    logger.info("Iniciando el lector de imágenes")
    lector = CargadorImagenes()

```

```
logger.info("Imágenes listas")
```

```
logger.info("Comienzo del entrenamiento")
gan.train(lector)
gan.serializar_red()
logger.info("Fin del entrenamiento")
```

B.1.6. lanzadera_produccion.py

```
import argparse
import pathlib

from cyclegan import CycleGAN
from utilidades import Utilidades, obtener_nombre_relativo_desde_string

if __name__ == "__main__":
    # Comandos entrada
    parser = argparse.ArgumentParser()
    parser.add_argument("-v", "--version",
                        help="Especifica el nombre de la versión "
                             "con la que se guardarán los archivos",
                        type=str,
                        default="version_por_defecto")
    parser.add_argument("-d", "--dataset",
                        help="Especifica el dataset a utilizar",
                        type=str,
                        default="monet2photo")
    parser.add_argument("-c", "--configuracion",
                        help="Especifica el fichero de configuración que se va a usar",
                        type=str,
                        default="configuracion_128.json")
    args = vars(parser.parse_args())

    assert pathlib.Path("../configuracion", args["configuracion"]).exists(), \
        "El archivo de configuración no existe"

    # inicializamos la clase de utilidades y la red
    utils = Utilidades(args["version"], args["dataset"], args["configuracion"])
    logger = utils.obtener_logger("lanzadera produccion")

    utils.asegurar_dataset()
    logger.info("Dataset en linea")

    gan = CycleGAN(restaurar=True, tipo_generador="unet")
    logger.info("Red neuronal lista")

    # listamos las imágenes que queremos traducir
    tipos_imagenes_admitidas = ['jpg', 'jpeg', 'bmp', 'png']

    imagenes_entrada_usuario = [(item.resolve() for item in
                                 pathlib.Path("../input").rglob("*")
                                 if item.parts[-1].split(".")[-1]
                                 in tipos_imagenes_admitidas)]

    imagenes_entrada_test_para_foto = utils.obtener_rutas_imagenes_test_pintor()
    imagenes_entrada_test_para_pintor = utils.obtener_rutas_imagenes_test_foto()

    # creamos las carpetas de salidas
    ruta_raiz_salida_usuario = (pathlib.Path("../output/")
                                / args["dataset"] / args["version"]).resolve()
    ruta_raiz_test_foto = (pathlib.Path("../output/"))
```

```

        / args["dataset"] / args["version"]
        / "test_foto").resolve()
ruta_raiz_test_pintor = (pathlib.Path("../output/")
        / args["dataset"] / args["version"]
        / "test_pintor").resolve()

ruta_raiz_salida_usuario.mkdir(parents=True, exist_ok=True)
ruta_raiz_test_foto.mkdir(parents=True, exist_ok=True)
ruta_raiz_test_pintor.mkdir(parents=True, exist_ok=True)

# especificamos las rutas de salida
imagenes_salida_usuario = [ruta_raiz_salida_usuario / imagen.parts[-1]
                           for imagen in imagenes_entrada_usuario]
imagenes_salida_test_para_foto = [ruta_raiz_test_foto
                                   / obtener_nombre_relativo_desde_string(imagen)
                                   for imagen in imagenes_entrada_test_para_foto]
imagenes_salida_test_para_pintor = [ruta_raiz_test_pintor
                                   / obtener_nombre_relativo_desde_string(imagen)
                                   for imagen in
                                   imagenes_entrada_test_para_pintor]

# calculamos
for entrada, salida in zip(imagenes_entrada_usuario, imagenes_salida_usuario):
    logger.info("Procesando " + salida.parts[-1])
    if salida.exists():
        continue
    imagen_cruda = gan.convertir_imagen(str(entrada), "pintor", 4)
    with open(salida, "wb") as archivo:
        archivo.write(imagen_cruda)

for entrada, salida in zip(imagenes_entrada_test_para_foto,
                           imagenes_salida_test_para_foto):
    logger.info("Procesando " + salida.parts[-1])
    if salida.exists():
        continue
    imagen_cruda = gan.convertir_imagen(str(entrada), "foto", 4)
    with open(salida, "wb") as archivo:
        archivo.write(imagen_cruda)

for entrada, salida in zip(imagenes_entrada_test_para_pintor,
                           imagenes_salida_test_para_pintor):
    logger.info("Procesando " + salida.parts[-1])
    if salida.exists():
        continue
    imagen_cruda = gan.convertir_imagen(str(entrada), "pintor", 4)
    with open(salida, "wb") as archivo:
        archivo.write(imagen_cruda)

```

B.1.7. procesado_imagenes.py

```

import base64
import imghdr

import numpy as np
import tensorflow as tf
import io
from PIL import Image

def normalizar_float(array_float):
    """Normaliza un array de floats al rango 0-1.
    :param array_float: el propio array a normalizar"""

```

```

salida = np.copy(array_float)
return np.clip(salida, 0, 255)

def comprueba_imagen(string_base64):
    """Comprueba si la imagen es válida y si lo es
    devuelve el tipo correspondiente.

    Parámetros:
        string_base64 : imagen en forma de string base64."""
    try:
        resultado = imghdr.what("ac", h=base64.b64decode(str(string_base64)))
    except:
        resultado = None
    return resultado

def imagen_a_base64_string(imagen, formato="JPEG"):
    """Convierte una imagen a un string_base64

    Parámetros:
        imagen: imagen en formato PIL Image.
        formato: parámetro opcional que
            indica la extensión de la imagen deseada."""
    buffer = io.BytesIO()
    imagen.save(buffer, format=formato)
    return base64.b64encode(buffer.getvalue())

def numpy_array_normalizado_a_imagen(numpy_array):
    """Convierte un array de valores entre 0 y 1 a una imagen
    en forma de objeto PIL.

    Parámetros:
        numpy_array: imagen en forma de numpy array.
        Se espera que los valores estén entre 0 y 1."""
    return Image.fromarray(float_array_a_int_array(numpy_array))

def float_array_a_int_array(numpy_array):
    """Convierte un array de valores en el rango 0-1
    a un array de enteros en el rango 0-255

    Parámetros:
        numpy_array: el propio array a convertir"""
    resultado = np.copy(numpy_array)
    if len(resultado.shape) == 4:
        resultado = np.squeeze(resultado, axis=0)
    resultado *= 255.0
    return (resultado.clip(0, 255) + 0.5).astype(np.uint8)

def numpy_array_a_imagen_PIL_bytes(imagen, extension):
    """Convierte un array en una imagen guardada como array de bytes.

    Parámetros:
        imagen: el propio array que contiene la imagen.
        Puede venir como un tensor en 3 o 4 dimensiones.
        extension: la extensión deseada en forma de string."""

    if len(tf.shape(imagen)) == 4:
        imagen = imagen.squeeze(axis=0)
        # Quitamos el primer eje para poder operar con ella

```

```

    imagen = (imagen.clip(0, 255)).astype(np.uint8)
    imagen_PIL = Image.fromarray(imagen)
    imagen_bytes = io.BytesIO()
    imagen_PIL.save(imagen_bytes, format=extension.upper())
    return imagen_bytes.getvalue()

def aumentar_resolucion(imagen, factor=4):
    """Reescala una imagen. Comparativa de métodos en
    https://www.tensorflow.org/api_docs/python/tf/image/resize

    Parámetros:
        imagen: el tensor que contiene la imagen.
        Puede venir en 3 o 4 dimensiones.
        factor: proporción en la que se amplia la imagen.
    """
    if len(tf.shape(imagen)) == 4:
        ancho = imagen.shape[1]
        alto = imagen.shape[2]
    else:
        ancho = imagen.shape[0]
        alto = imagen.shape[1]
    return tf.image.resize(imagen, [factor * ancho, factor * alto],
                          method=tf.image.ResizeMethod.BICUBIC).numpy()

def preprocessar_imagen(ruta, dimensiones):
    """Preprocesa una imagen y la devuelve como un tensor.

    Parámetros:
        ruta: ruta en disco de la imagen.
        dimensiones: dimensiones deseadas para la imagen."""
    imagen = tf.io.read_file(ruta) # Abrimos el archivo
    # Convertir el string a un tensor 3D uint8
    imagen = tf.image.decode_jpeg(imagen, channels=dimensiones[2], dct_method="INTEGER_ACCURATE")
    # Tenemos que cortar la imagen.
    # Desgraciadamente no se puede mantener el ratio del aspecto
    imagen = tf.image.resize(imagen, size=dimensiones[0:2],
                            method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
    return tf.image.convert_image_dtype(imagen / 255, tf.float32)

def preprocessar_imagen_individual(ruta, dimensiones):
    """Lee y preprocesa una imagen a partir de su ruta
    pero apto para usarse en producción.

    Parámetros:
        ruta: ruta en disco de la imagen.
        dimensiones: dimensiones deseadas para la imagen."""
    imagen = preprocessar_imagen(ruta, dimensiones)
    return tf.expand_dims(imagen, 0) # Añadimos la dimensión batch

```

B.1.8. singleton.py

```

class Singleton(type):
    """Clase que implementa el patrón singleton"""
    _instances = {}

    def __call__(cls, *args, **kwargs):
        if cls not in cls._instances:
            cls._instances[cls] = super(Singleton, cls).__call__(*args, **kwargs)
        return cls._instances[cls]

```

B.1.9. utilidades.py

```

import json
import logging
import pathlib
import platform
import shutil
import subprocess
import sys
from datetime import datetime

from tensorflow.keras.utils import get_file

from singleton import Singleton


def timestamp_fancy():
    """Timestamp con la fecha (dia-mes-año) y la hora (hora.minuto.segundo).
    Apto para directorios"""
    return timestamp().strftime("%d-%m-%Y %H.%M.%S")


def timestamp():
    """Obtener el timestamp actual"""
    return datetime.now()


def obtener_rutas_imagenes(ruta, expresion_regular='*.jpg'):
    """Obtiene las rutas de las imágenes de una carpeta"""
    return list(map(_ruta_a_string, list(ruta.glob(expresion_regular))))


def _ruta_a_string(path):
    """Convertir archivo del tipo Path a un string con la ruta absoluta."""
    return str(path.resolve())


def is_tool(name):
    """Check whether `name` is on PATH and marked as executable.
    Funcion recodiga de:
    https://stackoverflow.com/questions/11210104/
    check-if-a-program-exists-from-a-python-script"""

    return shutil.which(name) is not None


def gsutil_disponible():
    """Revisa si está disponible la herramienta gsutil,
    necesaria para las ejecuciones en GCP"""
    return is_tool("gsutil")


def obtener_nombre_relativo_desde_string(archivo):
    """Devuelve el nombre del archivo a partir de un string
    que contiene la ruta absoluta del mismo.

    Parámetros:
        archivo: string con la ruta de fichero"""
    return archivo.split("\\")[-1] \
        if platform.system() == "Windows" \
        else archivo.split("/")[-1]

```

```

class Utilidades(metaclass=Singleton):
    __slots__ = ["_version", "_dataset", "_ruta_logs", "_ruta_modelos",
                 "_ruta_raiz_dataset", "_ruta_tfcache", "_ruta_logs_entreno",
                 "_ruta_logs_test", "_ruta_modelo_modelos",
                 "_ruta_modelo_configuracion", "_archivo_dataset",
                 "_ruta_dataset", "_ruta_dataset_entreno_pintor",
                 "_ruta_logs_raiz", "_ruta_dataset_entreno_foto",
                 "_ruta_dataset_test_pintor", "_ruta_dataset_test_foto",
                 "_ruta_cache", "_tasa_aprendizaje",
                 "_lambda_reconstruccion", "_lambda_validacion",
                 "_lambda_identidad", "_dimensiones", "_epochs",
                 "_tamanio_buffer", "_tamanio_batch", "_filtros_generador",
                 "_filtros_discriminador", "_url_datasets",
                 "_url_api_aumento", "_gcp_bucket", "_imagen_pintor_muestra",
                 "_imagen_foto_muestra", "_ruta_archivo_muestra_pintor",
                 "_ruta_archivo_muestra_foto", "_mascara_logs", "_logger",
                 "_modelo_combinado", "_discriminador_pintor",
                 "_discriminador_foto", "_generador_pintor", "_generador_foto"]

    def __init__(self, version, dataset, archivo_configuracion):
        self._version = str(version)
        self._dataset = str(dataset)

        # Leemos el fichero json
        ruta_configuracion = pathlib.Path("../configuracion",
                                           archivo_configuracion).resolve()

        with open(ruta_configuracion) as archivo:
            datos_json = json.load(archivo)

        # Configuramos el resto de parámetros
        self._ruta_logs_raiz = pathlib.Path("../logs")
        self._ruta_logs = self._ruta_logs_raiz / self._dataset / self._version
        self._ruta_modelos = pathlib.Path("../modelos", self._dataset, self._version)
        self._ruta_raiz_dataset = pathlib.Path("../datasets")
        self._ruta_tfcache = pathlib.Path("../tfcache")

        self._ruta_logs_entreno = self._ruta_logs / "entreno"
        self._ruta_logs_test = self._ruta_logs / "test"

        self._ruta_modelo_modelos = self._ruta_modelos / "modelo"
        self._ruta_modelo_configuracion = self._ruta_modelos / "config"

        self._archivo_dataset = self._dataset + ".zip"
        self._ruta_dataset = self._ruta_raiz_dataset / self._dataset

        self._ruta_dataset_entreno_pintor = self._ruta_dataset / "trainA"
        self._ruta_dataset_entreno_foto = self._ruta_dataset / "trainB"
        self._ruta_dataset_test_pintor = self._ruta_dataset / "testA"
        self._ruta_dataset_test_foto = self._ruta_dataset / "testB"

        self._ruta_cache = self._ruta_tfcache / self._dataset / self._version
        # las caches son excluyentes entre iteraciones

        # Leemos los parámetros del modelo
        self._tasa_aprendizaje = float(datos_json["configuracion_modelo"]
                                         ["tasa_aprendizaje"])
        self._lambda_reconstruccion = float(datos_json["configuracion_modelo"]
                                              ["lambda_reconstruccion"])
        self._lambda_validacion = int(datos_json["configuracion_modelo"]
                                      ["lambda_validacion"])
        self._lambda_identidad = int(datos_json["configuracion_modelo"])

```

```

        ["lambda_identidad"])
_ancho = int(datos_json["configuracion_modelo"]["ancho"])
_alto = int(datos_json["configuracion_modelo"]["alto"])
_canales = int(datos_json["configuracion_modelo"]["canales"])
self._dimensiones = (_ancho, _alto, _canales)
self._epochs = int(datos_json["configuracion_modelo"]["epochs"])
self._tamanio_buffer = int(datos_json["configuracion_modelo"]
                           ["tamanio_buffer"])
self._tamanio_batch = int(datos_json["configuracion_modelo"]
                           ["tamanio_batch"])
self._filtros_generador = int(datos_json["configuracion_modelo"]
                               ["filtros_generador"])
self._filtros_discriminador = int(datos_json["configuracion_modelo"]
                                   ["filtros_discriminador"])

self._modelo_combinado = \
    self._ruta_modelo_modelos / "modelo_combinado.h5"
self._discriminador_pintor = \
    self._ruta_modelo_modelos / "discriminador_pintor.h5"
self._discriminador_foto = \
    self._ruta_modelo_modelos / "discriminador_foto.h5"
self._generador_pintor = \
    self._ruta_modelo_modelos / "generador_pintor.h5"
self._generador_foto = \
    self._ruta_modelo_modelos / "generador_foto.h5"

self._url_datasets = datos_json["url"]["datasets"] + self._archivo_dataset
self._url_api_aumento = datos_json["url"]["api_aumento"]

self._gcp_bucket = datos_json["gcp"]["bucket"]

self._imagen_pintor_muestra = \
    datos_json["dataset"][self._dataset]["imagen_pintor_muestra"]
self._imagen_foto_muestra = \
    datos_json["dataset"][self._dataset]["imagen_foto_muestra"]

self._ruta_archivo_muestra_pintor = \
    self._ruta_dataset_test_pintor / self._imagen_pintor_muestra
self._ruta_archivo_muestra_foto = \
    self._ruta_dataset_test_foto / self._imagen_foto_muestra

self._mascara_logs = datos_json["varios"]["mascara_logs"]

self._inicializar_directorios()
self._logger = self.obtener_logger("utilidades")

def _inicializar_directorios(self):
    """Crea los directorios a usar"""
    self._ruta_logs.mkdir(parents=True, exist_ok=True)
    self._ruta_logs_entreno.mkdir(parents=True, exist_ok=True)
    self._ruta_logs_test.mkdir(parents=True, exist_ok=True)
    self._ruta_modelos.mkdir(parents=True, exist_ok=True)
    self._ruta_modelo_configuracion.mkdir(parents=True, exist_ok=True)
    self._ruta_modelo_modelos.mkdir(parents=True, exist_ok=True)
    if self._ruta_tfcache.exists():
        shutil.rmtree(_ruta_a_string(self._ruta_tfcache),
                      ignore_errors=False, onerror=None)
    self._ruta_cache.mkdir(parents=True, exist_ok=True)

def asegurar_dataset(self):
    """Si no está el dataset en local, se descarga"""
    if not self._ruta_dataset.exists():

```

```

        self._logger.info("Descarga del dataset del repositorio")
        self._ruta_dataset.mkdir(parents=True)
        get_file(origin=self._url_datasets, fname=self._archivo_dataset,
                 extract=True, cache_dir=self._ruta_raiz_dataset,
                 cache_subdir="./")
    
```

```

        assert self._ruta_dataset_entreno_pintor.exists(), \
            "No existe el directorio de entreno pintor"
        assert self._ruta_dataset_entreno_foto.exists(), \
            "No existe el directorio de entreno real"
        assert self._ruta_dataset_test_pintor.exists(), \
            "No existe el directorio de test pintor"
        assert self._ruta_dataset_test_foto.exists(), \
            "No existe el directorio de test real"
        assert self._ruta_archivo_muestra_pintor.exists(), \
            "No existe la imagen de muestra del pintor"
        assert self._ruta_archivo_muestra_foto.exists(), \
            "No existe la imagen de muestra real"
    
```

```

def copiar_logs_gcp(self):
    """Ejecuta el proceso de copiar los logs al bucket de gcp."""
    subprocess.run(["gsutil", "-m", "rsync",
                   "-r", self._ruta_logs_raiz, self._gcp_bucket])

```

```

def obtener_logger(self, nombre):
    """Devuelve un logger que escribe tanto en fichero como en la salida estándar.

    Parámetros:

        nombre: nombre del logger a crear.
        Suele utilizarse el nombre de la clase/módulo."""
    logger = logging.getLogger(nombre)
    formateador = logging.Formatter(self._mascara_logs)
    logger.setLevel(logging.INFO)

    manejador_salida_estandar = logging.StreamHandler(sys.stdout)
    manejador_salida_estandar.setFormatter(formateador)
    manejador_archivo = logging.FileHandler(self._obtener_archivo_logger(nombre))
    manejador_archivo.setFormatter(formateador)

    logger.addHandler(manejador_archivo)
    logger.addHandler(manejador_salida_estandar)

    return logger

```

```

# Funciones para obtener los atributos privados
def obtener_rutas_imagenes_entreno_pintor(self):
    return obtener_rutas_imagenes(self._ruta_dataset_entreno_pintor)

def obtener_rutas_imagenes_entreno_foto(self):
    return obtener_rutas_imagenes(self._ruta_dataset_entreno_foto)

def obtener_rutas_imagenes_test_pintor(self):
    return obtener_rutas_imagenes(self._ruta_dataset_test_pintor)

def obtener_rutas_imagenes_test_foto(self):
    return obtener_rutas_imagenes(self._ruta_dataset_test_foto)

def obtener_ruta_logs(self):
    return _ruta_a_string(self._ruta_logs)

def obtener_ruta_logs_entreno(self):

```

```

        return _ruta_a_string(self._ruta_logs_entreno)

def obtener_ruta_logs_test(self):
    return _ruta_a_string(self._ruta_logs_test)

def obtener_archivo_muestra_pintor(self):
    return _ruta_a_string(self._ruta_archivo_muestra_pintor)

def obtener_archivo_muestra_foto(self):
    return _ruta_a_string(self._ruta_archivo_muestra_foto)

def obtener_archivo_cache(self, nombre):
    return _ruta_a_string(self._ruta_cache / (nombre + ".tfcache"))

def obtener_ruta_archivo_modelo_parametros(self):
    return _ruta_a_string(self._ruta_modelo_configuracion
                           / "parametros.pkl")

def obtener_ruta_archivo_modelo_objeto(self):
    return _ruta_a_string(self._ruta_modelo_configuracion
                           / "cyclegan.pkl")

def obtener_ruta_archivo_modelo_esquema(self):
    return _ruta_a_string(self._ruta_modelo_configuracion
                           / "modelo_combinado.png")

def obtener_ruta_archivo_discriminador_pintor_esquema(self):
    return _ruta_a_string(self._ruta_modelo_configuracion
                           / "discriminador_pintor.png")

def obtener_ruta_archivo_discriminador_foto_esquema(self):
    return _ruta_a_string(self._ruta_modelo_configuracion
                           / "discriminador_foto.png")

def obtener_ruta_archivo_generador_pintor_esquema(self):
    return _ruta_a_string(self._ruta_modelo_configuracion
                           / "generador_pintor.png")

def obtener_ruta_archivo_generador_foto_esquema(self):
    return _ruta_a_string(self._ruta_modelo_configuracion
                           / "generador_foto.png")

def obtener_ultimos_pesos(self):
    lista_pesos = list(pathlib.Path(self._ruta_modelo_modelos).
                        rglob("pesos-*.*h5")) # obtenemos la lista de pesos
    if lista_pesos:
        maximo = max(list(map(lambda x:
                               int(str(x).split("-")[-1].split(".")[0]),
                               lista_pesos)))
        # procesamos (si hay) el nombre de fichero:
        # nos quedamos con la parte dcha del guion y
        # la izquierda del punto, es decir
        # el numero de epoch. Obtenemos el maximo
        return _ruta_a_string(self._ruta_modelo_modelos
                               / ("pesos-" + str(maximo) + ".h5")), maximo
    else:
        return None, 1

def existen_ficheros_modelo(self):
    return self._modelo_combinado.exists() \
           and self._discriminador_pintor.exists() \
           and self._discriminador_foto.exists() \

```

```

        and self._generador_pintor.exists() \
        and self._generador_foto.exists()

def obtener_ruta_fichero_modelo(self):
    return _ruta_a_string(self._modelo_combinado)

def obtener_ruta_fichero_discriminador_pintor(self):
    return _ruta_a_string(self._discriminador_pintor)

def obtener_ruta_fichero_discriminador_foto(self):
    return _ruta_a_string(self._discriminador_foto)

def obtener_ruta_fichero_generador_pintor(self):
    return _ruta_a_string(self._generador_pintor)

def obtener_ruta_fichero_generador_foto(self):
    return _ruta_a_string(self._generador_foto)

def obtener_ruta_fichero_modelo_por_epoch(self, epoch):
    return _ruta_a_string(self._ruta_modelo_modelos
                           / ("pesos-" + str(epoch) + ".h5"))

def _obtener_archivo_logger(self, nombre):
    return _ruta_a_string(self._ruta_logs / (nombre + ".log"))

def obtener_dimensiones(self):
    return self._dimensiones

def obtener_tamano_batch(self):
    return self._tamano_batch

def obtener_tamano_buffer(self):
    return self._tamano_buffer

def obtener_tasa_aprendizaje(self):
    return self._tasa_aprendizaje

def obtener_lambda_reconstruccion(self):
    return self._lambda_reconstruccion

def obtener_lambda_validacion(self):
    return self._lambda_validacion

def obtener_lambda_identidad(self):
    return self._lambda_identidad

def obtener_filtros_generador(self):
    return self._filtros_generador

def obtener_filtros_discriminador(self):
    return self._filtros_discriminador

def obtener_epochs(self):
    return self._epochs

def obtener_url_api_aumento(self):
    return self._url_api_aumento

```

B.1.10. requirements.txt

```

keras==2.3.1
tensorflow==2.2.1

```

```
tensorflow_datasets
git+https://github.com/tensorflow/examples.git
git+https://www.github.com/keras-team/keras-contrib.git
numpy==1.18.5
matplotlib
```

B.2. Archivos de configuración del sistema principal

B.2.1. configuracion_128.json

```
{
  "configuracion_modelo":
  {
    "tasa_aprendizaje": "0.0002",
    "lambda_reconstruccion": "10.0",
    "lambda_validacion": "1",
    "lambda_identidad": "2",
    "ancho": "128",
    "alto": "128",
    "canales": "3",
    "epochs": "200",
    "tamanio_buffer": "1000",
    "tamanio_batch": "1",
    "filtros_generador": "32",
    "filtros_discriminador": "32"
  },
  "url":
  {
    "api_aumento": "http://localhost:5000/TFG-Computadores/api-aumento/aumento",
    "datasets": "https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/"
  },
  "gcp":
  {
    "bucket": "gs://tfgr-impressionismo/"
  },
  "dataset":
  {
    "monet2photo":
    {
      "imagen_pintor_muestra": "00360.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "cezanne2photo":
    {
      "imagen_pintor_muestra": "00100.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "ukiyoe2photo":
    {
      "imagen_pintor_muestra": "01214.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "vangogh2photo":
    {
      "imagen_pintor_muestra": "00021.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    }
  },
  "varios":
  {
    "mascara_logs" : "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
  }
}
```

B.2.2. configuracion_256.json

```
{
  "configuracion_modelo":
  {
    "tasa_aprendizaje": "0.0002",
    "lambda_reconstruccion": "10.0",
    "lambda_validacion": "1",
    "lambda_identidad": "2",
    "ancho": "256",
    "alto": "256",
    "canales": "3",
    "epochs": "200",
    "tamanio_buffer": "1000",
    "tamanio_batch": "1",
    "filtros_generador": "64",
    "filtros_discriminador": "64"
  },
  "url":
  {
    "api_aumento": "http://localhost:5000/TFG-Computadores/api-aumento/aumento",
    "datasets": "https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/"
  },
  "gcp":
  {
    "bucket": "gs://tfg-impresionismo/"
  },
  "dataset":
  {
    "monet2photo":
    {
      "imagen_pintor_muestra": "00360.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "cezanne2photo":
    {
      "imagen_pintor_muestra": "00100.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "ukiyoe2photo":
    {
      "imagen_pintor_muestra": "01214.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "vangogh2photo":
    {
      "imagen_pintor_muestra": "00021.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    }
  },
  "varios":
  {
    "mascara_logs" : "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
  }
}
```

B.2.3. configuracion_512.json

```
{
  "configuracion_modelo":
  {
    "tasa_aprendizaje": "0.0002",
    "lambda_reconstruccion": "10.0",
    "lambda_validacion": "1",
    "lambda_identidad": "2",
    "ancho": "512",
    "alto": "512",
    "canales": "3",
    "epochs": "200",
    "tamanio_buffer": "1000",
    "tamanio_batch": "1",
    "filtros_generador": "64",
    "filtros_discriminador": "64"
  },
  "url":
  {
    "api_aumento": "http://localhost:5000/TFG-Computadores/api-aumento/aumento",
    "datasets": "https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/"
  },
  "gcp":
  {
    "bucket": "gs://tfg-impresionismo/"
  },
  "dataset":
  {
    "monet2photo":
    {
      "imagen_pintor_muestra": "00360.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "cezanne2photo":
    {
      "imagen_pintor_muestra": "00100.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "ukiyoe2photo":
    {
      "imagen_pintor_muestra": "01214.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "vangogh2photo":
    {
      "imagen_pintor_muestra": "00021.jpg",
      "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    }
  },
  "varios":
  {
    "mascara_logs" : "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
  }
}
```

```

    "lambda_reconstruccion": "10.0",
    "lambda_validacion": "1",
    "lambda_identidad": "2",
    "ancho": "512",
    "alto": "512",
    "canales": "3",
    "epochs": "200",
    "tamanio_buffer": "1000",
    "tamanio_batch": "1",
    "filtros_generador": "128",
    "filtros_discriminador": "128"
},
"url":
{
    "api_aumento": "http://localhost:5000/TFG-Computadores/api-aumento/aumento",
    "datasets": "https://people.eecs.berkeley.edu/~taesung_park/CycleGAN/datasets/"
},
"gcp":
{
    "bucket": "gs://tfwg-impresionismo/"
},
"dataset":
{
    "monet2photo":
    {
        "imagen_pintor_muestra": "00360.jpg",
        "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "cezanne2photo":
    {
        "imagen_pintor_muestra": "00100.jpg",
        "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "ukiyoe2photo":
    {
        "imagen_pintor_muestra": "01214.jpg",
        "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    },
    "vangogh2photo":
    {
        "imagen_pintor_muestra": "00021.jpg",
        "imagen_foto_muestra": "2014-08-24_16_41_27.jpg"
    }
},
"varios":
{
    "mascara_logs" : "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
}
}

```

B.3. Sistema ampliador

Al igual que el sistema principal, el código está disponible bajo el repositorio *EnhanceNet-TFG*, accesible en la url <https://github.com/VicDominguez/TFG-Computadores-Principal>.

Ahí se pueden encontrar además los pesos del modelo utilizado en este documento.

B.3.1. controlador_modelo.py

```
from modelo import inferencia
from procesado import *

def _realizar_ampliacion(image):
    return inferencia(preprocesar_imagen(image))

def ampliar_desde_archivo_base64_a_archivo_base64(ruta_entrada, ruta_salida):
    imagen_entrada = abrir_imagen_desde_archivo_base64(ruta_entrada)
    imagen_ampliada = _realizar_ampliacion(imagen_entrada)
    guardar_imagen_a_archivo_base64(imagen_ampliada, ruta_salida)

def ampliar_desde_archivo_base64_a_archivo_imagen(ruta_entrada, ruta_salida):
    imagen_entrada = abrir_imagen_desde_archivo_base64(ruta_entrada)
    imagen_ampliada = _realizar_ampliacion(imagen_entrada)
    guardar_imagen_archivo_imagen(imagen_ampliada, ruta_salida)

def ampliar_desde_archivo_imagen_a_archivo_imagen(ruta_entrada, ruta_salida):
    imagen_entrada = abrir_imagen_desde_archivo_imagen(ruta_entrada)
    imagen_ampliada = _realizar_ampliacion(imagen_entrada)
    guardar_imagen_archivo_imagen(imagen_ampliada, ruta_salida)

def ampliar_desde_archivo_imagen_a_archivo_base64(ruta_entrada, ruta_salida):
    imagen_entrada = abrir_imagen_desde_archivo_imagen(ruta_entrada)
    imagen_ampliada = _realizar_ampliacion(imagen_entrada)
    guardar_imagen_a_archivo_base64(imagen_ampliada, ruta_salida)

def ampliar_desde_string_base64_a_string_base64(string_base64, formato_imagen="JPEG"):
    imagen_entrada = abrir_imagen_desde_string_base64(string_base64)
    imagen_ampliada = _realizar_ampliacion(imagen_entrada)
    return guardar_imagen_a_string_base64(imagen_ampliada, formato_imagen=formato_imagen)
```

B.3.2. modelo.py

```
import tensorflow as tf
from utils import PER_CHANNEL_MEANS, obtener_ruta_pesos

# Silenciamos los errores de depreciación de tf.contrib.layers.convolution2d
tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

def _conv(h, n=64):
    """Capa de convolución 2D"""

    # Definimos la capa de convolución
    with tf.variable_scope('conv'):
        w = tf.get_variable('w', [3, 3, h.get_shape()[-1], n],
                            initializer=tf.truncated_normal_initializer(stddev=0.02))
        b = tf.get_variable('b', [n], initializer=tf.constant_initializer(0.0))
```

```

    return tf.contrib.layers.convolution2d(h, n, kernel_size=3,
                                         stride=1, padding='SAME',
                                         activation_fn=None)

def _relu(h):
    """Función de activación relú"""
    return tf.nn.relu(h)

def _ampliar(h):
    """Capa para ampliar la imagen en un factor de 2"""
    return tf.image.resize(h, [2 * tf.shape(h)[1], 2 * tf.shape(h)[2]],
                          method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)

def _bloque_residual(h, subcapas):
    """Crea el bloque residual a partir de las subcapas que entran por parámetro"""
    htemp = h
    for subcapa in subcapas:
        h = subcapa[0](h, *subcapa[1:])
    h += htemp
    return h

def _crear_red(nombre, capas):
    """Crea la red a partir de una lista de capas"""
    h = capas[0]
    with tf.compat.v1.variable_scope(nombre, reuse=False) as scope:
        for capa in capas[1:]:
            h = capa[0](h, *capa[1:])
    return h

def inferencia(imagen):
    """Crea la red, carga los pesos y amplia la imagen de entrada en un factor de 4"""
    tamanio_imagen = imagen.shape[1:]
    # Creamos el placeholder de la entrada del grafo de tensorflow
    entrada_placeholder = tf.compat.v1.placeholder(tf.float32,
                                                   [1,
                                                    tamanio_imagen[0],
                                                    tamanio_imagen[1],
                                                    tamanio_imagen[2]])

    # Definimos la red
    bloque_r = [_bloque_residual, [[_conv], [_relu], [_conv]]]
    salida_estimada = _crear_red('generator',
                                 [entrada_placeholder,
                                  [_conv], [_relu],
                                  bloque_r, bloque_r, bloque_r, bloque_r, bloque_r,
                                  bloque_r, bloque_r, bloque_r, bloque_r, bloque_r,
                                  [_ampliar], [_conv], [_relu],
                                  [_ampliar], [_conv], [_relu],
                                  [_conv], [_relu],
                                  [_conv, 3]]])
    # Definimos la imagen bicubica y configuramos la salida
    imagen_bicubica = tf.image.resize_images(entrada_placeholder,
                                              [4 * tamanio_imagen[0],
                                               4 * tamanio_imagen[1]],
                                              method=tf.image.ResizeMethod.BICUBIC)
    salida_estimada += imagen_bicubica + PER_CHANNEL_MEANS
    # Creamos la sesión, cargamos los pesos y ejecutamos
    sess = tf.compat.v1.InteractiveSession()
    tf.compat.v1.train.Saver().restore(sess, obtener_ruta_pesos())

```

```

salida = sess.run(salida_estimada, feed_dict={entrada_placeholder:
                                              imagen - PER_CHANNEL_MEANS})
sess.close()
tf.compat.v1.reset_default_graph()
return salida[0]

```

B.3.3. principal.py

```

import base64

from controlador_modelo import *
from flask import Flask, jsonify, request, abort, make_response
import imghdr

app = Flask(__name__)

def comprueba_imagen(string_base64):
    """Comprueba si la imagen es válida y devuelve el tipo correspondiente"""
    try:
        resultado = imghdr.what("ac", h=base64.b64decode(str(string_base64)))
    except:
        resultado = None
    return resultado

"""Manejador para el error 404"""

@app.errorhandler(404)
def not_found(error):
    return make_response(jsonify({'error': 'Not found'}), 404)

"""Función que llama al controlador del modelo revisando los parámetros.
Necesita una petición con el campo imagen y la imagen codificada en base64
y evuelve la imagen codificada en base64 en el campo imagen_ampliada"""

@app.route('/TFG-Computadores/api-aumento/aumento', methods=['POST'])
def realizar_aumento():
    if not request.json or 'imagen' not in request.json:
        abort(402)
    imagen = request.json['imagen']
    tipo_imagen = comprueba_imagen(imagen)
    if tipo_imagen is not None:
        resultado = ampliar_desde_string_base64_a_string_base64(
            imagen, formato_imagen=tipo_imagen)
        return jsonify({'imagen_ampliada': str(resultado)}), 200
    else:
        abort(400)

if __name__ == "__main__":
    app.run(debug=True, host="0.0.0.0")

```

B.3.4. procesado.py

```

import base64
from PIL import Image
import io

```

```
import numpy as np

"""Funciones para procesar la entrada"""

def abrir_imagen_desde_archivo_base64(ruta):
    with open(ruta, "r") as archivo:
        string_base64 = archivo.read()
    return abrir_imagen_desde_string_base64(string_base64)

def abrir_imagen_desde_string_base64(string_base64):
    imagen = base64.b64decode(str(string_base64))
    return Image.open(io.BytesIO(imagen)).convert('RGB')

def abrir_imagen_desde_archivo_imagen(ruta):
    return Image.open(ruta).convert('RGB')

"""Preprocesado"""

def preprocessar_imagen(imagen):
    imagen = np.array(imagen) / 255
    # Añadimos el cuarto eje debido a que tensorflow necesita un tensor 4D
    return np.expand_dims(imagen, axis=0)

"""Procesado salida"""

def _numpy_array_a_imagen(imagen):
    # Multiplicamos el rango por 255 y lo cortamos a 0-255
    # para pasarlo a entero de 1 byte sin signo
    imagen *= 255.0
    imagen = (imagen.clip(0, 255) + 0.5).astype(np.uint8)
    if len(np.shape(imagen)) > 2 and np.shape(imagen)[2] == 1:
        imagen = np.reshape(imagen, (np.shape(imagen)[0], np.shape(imagen)[1]))
    return Image.fromarray(imagen)

def guardar_imagen_archivo_imagen(imagen, ruta):
    im = _numpy_array_a_imagen(imagen)
    im.save(ruta)

def guardar_imagen_a_archivo_base64(imagen, ruta, formato_imagen="JPEG"):
    imagen_base64_string = guardar_imagen_a_string_base64(imagen, formato_imagen)
    with open(ruta, "w") as archivo:
        archivo.write(str(imagen_base64_string, "utf-8"))

def guardar_imagen_a_string_base64(imagen, formato_imagen="JPEG"):
    imagen = _numpy_array_a_imagen(imagen)
    imagen_bytes = io.BytesIO()
    imagen.save(imagen_bytes, format=formato_imagen)
    imagen_bytes = imagen_bytes.getvalue()
    return str(base64.b64encode(imagen_bytes), "utf-8")
```

B.3.5. utils.py

```
import numpy as np
import pathlib

PER_CHANNEL_MEANS = np.array([0.47614917, 0.45001204, 0.40904046])
_ruta_pesos = pathlib.Path("./weights")

_carpeta_entrada = pathlib.Path("../input")
_carpeta_salida = pathlib.Path("../output")

def _ruta_a_string(path):
    """Convert Path object to absolute path string."""
    return str(path.resolve())

def obtener_ruta_pesos():
    return _ruta_a_string(_ruta_pesos)

def obtener_ruta_archivo_entrada(archivo):
    """Devuelve la ruta para un nombre de archivo de entrada"""
    return (_carpeta_entrada / archivo).resolve()

def obtener_ruta_archivo_salida(archivo):
    """Devuelve la ruta para un nombre de archivo de salida"""
    return (_carpeta_salida / archivo).absolute()
```

B.3.6. requirements.txt

```
tensorflow==1.14.0
pillow
flask
```