Graduada en Ingeniería del Software

Restauración Digital de Arte y Patrimonio Cultural mediante Modelos y Técnicas de Aprendizaje Profundo

Realizado por
Alba de la Torre Segato

Tutorizado por
Juan Miguel Ortiz de Lazcano Lobato

Departamento
Lenguajes y Ciencias de la Computación
UNIVERSIDAD DE MÁLAGA

MÁLAGA, junio de 2025

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
GRADUADA EN INGENIERÍA DEL SOFTWARE

Restauración Digital de Arte y Patrimonio Cultural mediante Modelos y Técnicas de Aprendizaje Profundo

Digital Restoration of Art and Cultural Heritage through Deep Learning Models and Techniques

Realizado por
Alba de la Torre Segato

Tutorizado por
Juan Miguel Ortiz de Lazcano Lobato

Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, JUNIO DE 2025

Fecha defensa: Julio de 2025

Resumen

El objetivo de este trabajo es desarrollar un modelo basado en aprendizaje profundo capaz de restaurar obras de arte dañadas mediante técnicas de *inpainting*, es decir, el completado de regiones deterioradas de una imagen para integrarlas de forma consistente con el resto. Esto presenta dos grandes retos: la reconstrucción de estructuras globales coherentes y la preservación de las texturas locales. Para abordarlos se han utilizado dos métodos diferentes.

En primer lugar se implementó un *Autoencoder*[1] para capturar las características relevantes de los estilos pictóricos y reconstruir las zonas con desperfectos. Se exploraron distintas arquitecturas y configuraciones de entrenamiento, destacando la integración de dicho *Autoencoder* como componente generador de una *Generative Adversarial Network (GAN)*[2], entrenándose el sistema hasta que el discriminador no pudo diferenciar entre imágenes reales y las reconstruidas por el *GAN*. Ambos enfoques se refinaron mediante sucesivas pruebas y ajuste de hiperparámetros.

Por otro lado, también se aborda el problema utilizando el modelo neuronal profundo conocido como *Image Completion Transformer*[3], diseñado para aprovechar la atención global de los *Transformers* junto a la potencia de las redes convolucionales, que son capaces de generar texturas detalladas y realistas.

Ambos enfoques se evalúan cuantitativa y cualitativamente mediante comparaciones visuales y concluyen, tras analizar los resultados, que el modelo basado en *ICT* produce reparaciones de las obras con una mayor coherencia visual.

Palabras clave: aprendizaje profundo, *inpainting, GAN, Transformers*, restauración digital.

[1] Autoencoder: arquitectura compuesta por un codificador y un decodificador.

[2] *GAN* (Generative Adversarial Network): arquitectura compuesta por un generador y un discriminador entrenados como adversarios.

[3] ICT (Image Completion Transformer)[1].

Abstract

The aim of this work is to develop a deep-learning–based model capable of restoring damaged works of art through inpainting, that is, completing deteriorated regions so that they integrate consistently with the rest of the image. This poses two major challenges: the reconstruction of coherent global structures and the preservation of local textures. To address them, two complementary methods have been employed.

First, an autoencoder[4] was implemented to learn the pictorial styles' most relevant features and directly reconstruct the damaged areas. Various network architectures and training configurations were explored, notably integrating this autoencoder as the generator within a Generative Adversarial Network (GAN)[5]. The system was trained until the discriminator could no longer distinguish between real images and those reconstructed by the GAN. Both the standalone autoencoder and the GAN-based variant were refined through iterative experiments and careful hyperparameter tuning.

Secondly, we adopted the Image Completion Transformer[6], specifically designed to take advantage of the global attention mechanism of transformers together with the powerful texture-modeling capacity of convolutional neural networks, generating detailed and realistic textures.

Both approaches were evaluated quantitatively and qualitatively via visual comparisons. After analyzing the results, we conclude that the ICT-based model achieves superior visual coherence by effectively connecting distant regions and minimizing perceptual error.

Keywords: deep learning, inpainting, GAN, transformers, digital restoration.

[4]Autoencoder: architecture composed of an encoder and a decoder.

[5]GAN (Generative Adversarial Network): architecture composed of a generator and a discriminator trained adversarially.

[6]ICT (Image Completion Transformer)[1].

Índice

1. Introducción **7**

1.1. Motivación 7
1.2. Estado del arte 8
1.3. Objetivos 8
1.4. Tecnologías usadas 9
1.5. Metodología de trabajo 10
1.6. Estructura del documento 10

2. Conceptos Preliminares **11**

2.1. La neurona artificial clásica 11
2.2. La neurona convolucional 14
2.3. De neuronas a redes 16
2.4. Entrenamiento de redes neuronales 18
2.5. Tipos de capas en una red 21
2.6. Arquitecturas profundas especializadas 23
2.7. Autoencoders 23
2.8. GANs 25
2.9. Transformers 27
2.10. Medidas de entrenamiento y rendimiento 31
2.11. Contraste de hipótesis 33
2.12. El ruido *Perlin* 35

3. Diseño del Sistema **39**

3.1. Creación del *dataset* de entrenamiento 39
3.2. Primera aproximación: Autoencoders 41
3.3. Segunda aproximación: *GANs* 47
3.4. El modelo ICT 51

4. Análisis de los resultados **55**

4.1. Estudio cualitativo 55

4.2. Estudio cuantitativo 59

5. Uso del Sistema **63**

6. Conclusiones y Líneas Futuras **67**

6.1. Conclusiones 67

6.2. Líneas Futuras 68

Apéndice A. Instalación **75**

1 Introducción

1.1. Motivación

Este Trabajo de Fin de Grado se encuentra en la intersección entre el ámbito en constante evolución de la inteligencia artificial y la conservación del patrimonio cultural. La motivación para desarrollar este trabajo nace tras la reconstrucción de la obra ”La ronda de noche” del artista Rembrandt, realizada mediante un *pipeline*[7] basado en redes neuronales profundas [2]. Gracias a este enfoque, la obra pudo volver a ser admirada en su totalidad después de 300 años. Este logro puso en evidencia el gran potencial de las redes neuronales para la conservación y recuperación del patrimonio artístico.

La preservación y restauración de obras de arte y patrimonio cultural son esenciales para garantizar que la herencia cultural de una sociedad no se pierda con el paso del tiempo. Estas piezas son parte de lo que somos y también de quiénes fuimos, por lo que preservarlas es un deber con nuestra historia y un compromiso con las generaciones futuras, que merecen conocer y disfrutar de este patrimonio.

Sin embargo, a lo largo de la historia, la conservación de estas piezas se ha visto amenazada, poniendo en riesgo la preservación de este importante legado. Ejemplos como el terremoto de 1755 de Lisboa, donde muchas de las obras literarias guardadas en la Biblioteca Nacional de Portugal fueron destruidas, el incendio del Alcázar Real de Madrid en 1734, en el cual se perdieron obras de valor inestimable de artistas como Velázquez, Rubens o El Greco, o la destrucción casi total de la ciudad culturalmente valiosa de Dresde durante los bombardeos de la Segunda Guerra Mundial, demuestran la fragilidad del legado humano.

Estos eventos, junto a otros factores como el deterioro provocado por los años han causado daños irreparables en estas piezas, o incluso la pérdida total de ellas. En este contexto, surge

[7]Pipeline: Serie de pasos que optimizan el proceso de entrega de software.

la necesidad de explorar soluciones digitales que complementen las técnicas físicas de restauración, aprovechando la capacidad de los modelos de aprendizaje profundo para comprender patrones visuales y encontrar similitudes con una precisión superior a la humana.

Los modelos de aprendizaje profundo como los Autoencoders [3] [4] o el modelo *ICT* [1] son capaces de analizar la imagen completa y entender sus patrones globales, en lugar de procesarla por partes. Al entrenarlos con imágenes similares o del mismo estilo de la que queremos conservar, deberían ser capaces de aprender la información significativa que les permita completar las áreas faltantes basándose en el contexto visual de la imagen de manera coherente, lo que facilitaría la preservación del estilo artístico y la estructura original.

1.2. Estado del arte

En los últimos años, el campo de la reconstrucción de imágenes ha experimentado una transformación radical gracias al auge del aprendizaje profundo. Entre los modelos más influyentes se encuentran los *Context Encoders* [5], pioneros en el uso de redes convolucionales profundas para aprender a rellenar regiones ausentes. Otro avance relevante fue *EdgeConnect* [6], que planteó una estrategia en dos etapas, primero una reconstrucción de los bordes perdidos, seguida de un relleno de textura siguiendo los bordes generados.

Más recientemente, han ganado protagonismo las arquitecturas basadas en *Transformers*, que superan las limitaciones locales de las redes convolucionales. El modelo *ICT (Image Completion Transformer)*, que se usará en este trabajo destaca por conectar correctamente zonas alejadas de la imagen y conservar el contenido visual de forma coherente[1]. Por otro lado, modelos como *RePaint* [7] o *Palette* del equipo de *Google Research* [8] utilizan técnicas para generar múltiples muestras plausibles, alcanzando resultados altamente realistas y siendo especialmente robustos cuando las zonas faltantes son grandes o tienen formas irregulares. En conjunto, estos modelos representan el estado del arte por su capacidad de generar reconstrucciones visualmente coherentes que mantienen la estructura semántica de la imagen original, lo cual es especialmente relevante en contextos de restauración de patrimonio cultural.

1.3. Objetivos

Los objetivos de este trabajo son los siguientes:

1. Desarrollar un modelo neuronal profundo capaz de reconstruir los fragmentos faltantes en obras de arte y comparar los resultados con otros modelos como el *ICT*.

2. Estudiar cómo el ajuste de los hiperparámetros del modelo afecta a la reconstrucción de obras de distintos estilos artísticos.

3. Desarrollar un sistema capaz de generar daños digitales en obras de arte de forma realista, simulando deterioros comunes con el fin de entrenar y evaluar el modelo de reconstrucción con mayor fidelidad.

4. Analizar el desempeño del sistema con métricas cuantitativas.

5. Desarrollar una interfaz web para facilitar el uso del modelo.

1.4. Tecnologías usadas

Las tecnologías empleadas en el desarrollo de este trabajo incluyen principalmente el lenguaje de programación *Python*, ampliamente reconocido como la opción por excelencia en el ámbito de la inteligencia artificial, gracias a su facilidad de uso y gran potencia.

Las bibliotecas usadas han sido principalmente *Keras* [9] y *Tensorflow* [10]. *Keras* es una herramienta que facilita notablemente la creación de modelos neuronales profundos gracias a su simplicidad y claridad en la estructura del código. Una de las principales fortalezas de *Keras* se encuentra en su integración dentro de *TensorFlow*, una de las bibliotecas más potentes para computación numérica, especializada en cálculos con tensores de varias dimensiones. Esta integración permite que los modelos creados con *Keras* aprovechen la eficiencia, escalabilidad y velocidad de procesamiento de *TensorFlow*, especialmente en operaciones complejas y grandes volúmenes de datos.

Estas bibliotecas también están específicamente optimizadas para funcionar sobre unidades de procesamiento gráfico (GPU) con arquitectura CUDA. En este proyecto, el entrenamiento se ha llevado a cabo en un servidor de la Universidad de Málaga que cuenta con GPU con soporte CUDA, lo que ha facilitado el uso eficiente de estas librerías.

En resumen, *Keras* y *TensorFlow* forman una combinación ideal que simplifica enormemente el proceso de desarrollo y entrenamiento de modelos de aprendizaje profundo, facilitando

tanto la fase experimental como su posterior implementación en producción. Finalmente, la interfaz web ha sido desarrollada utilizando el *framework Flask*.

1.5. Metodología de trabajo

Para el desarrollo del trabajo se utilizó la metodología ágil *Scrum*, junto con los principios *PMBOK* para mejorar la organización de cada *sprint*. Este enfoque permitió un desarrollo flexible e iterativo, permitiendo cambios de acuerdo con los resultados de cada *sprint*.

El uso de *PMBOK* permitió estructurar cada *sprint* con una planificación clara, asegurando la alineación del proyecto con los objetivos del TFG, mientras que *Scrum* proporcionó la flexibilidad necesaria para realizar ajustes y mejoras continuas a medida que avanzaba el desarrollo. Todos los cambios y avances significativos han sido supervisados por el tutor, realizando los ajustes pertinentes antes de seguir con el proyecto.

1.6. Estructura del documento

Este documento se estructura de la siguiente manera:

1. Introducción: Presentación del contexto y la motivación tras el trabajo.
2. Conceptos preliminares: Incluye la base teórica del trabajo y fundamentos del aprendizaje profundo.
3. Diseño del sistema: Detalla las decisiones tomadas durante la realización de este trabajo y su justificación.
4. Análisis de los resultados: Se analizan los resultados de los modelos mediante métricas cuantitativas.
5. Uso del sistema: Explicación del funcionamiento de la interfaz final y de la instalación y puesta en marcha del modelo.
6. Conclusiones y líneas futuras del trabajo: Resumen de los hallazgos encontrados y presentación de las posibles mejoras y ampliaciones del trabajo.
7. Bibliografía: Recoge todas las referencias citadas durante el trabajo.

2

Conceptos Preliminares

A continuación se detalla el marco teórico tras los modelos utilizados durante el trabajo.

2.1. La neurona artificial clásica

Los modelos utilizados son cada uno un tipo distinto de red neuronal, que es una estructura inspirada en las redes neuronales biológicas de los cerebros humanos. Consta de capas de neuronas, interconectadas entre sí.

Una neurona simple recibe una entrada x_i y unos pesos sinápticos w_i y produce una salida $y = \sigma(z)$, donde σ es la función de activación y z representa el potencial sináptico calculado como se indica a continuación:

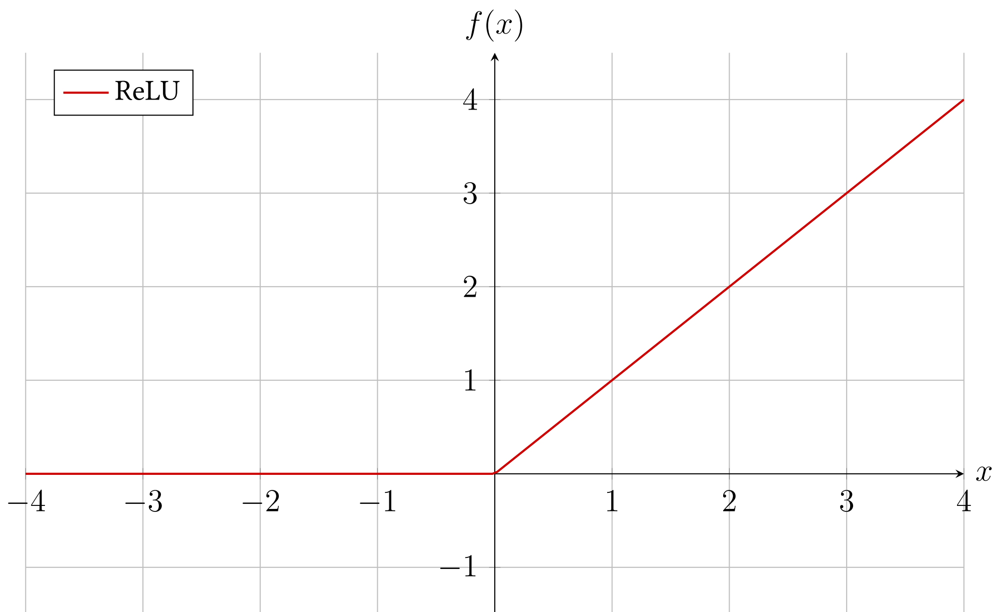
$$z \;=\; \sum_{i=1}^{n} w_i\, x_i \;+\; b, \text{donde } b \text{ modela el sesgo cognitivo.} \tag{1}$$

Cada neurona recibe un vector de entrada cuyas componentes representan la información proveniente de otras neuronas a las que está conectada. A cada una de estas entradas se le asocia un peso, que actúa como un factor de ponderación que regula la influencia de dicha información en las siguientes neuronas. Durante el proceso de aprendizaje, estos pesos sinápticos se ajustan en función de los datos de entrenamiento, de modo que terminan reflejando lo que la neurona ha aprendido.

La salida y de la neurona se obtiene aplicando una función de activación al potencial sináptico.

La función de activación es el componente de un modelo que introduce no linealidad al mismo, permitiendo aprender patrones más complejos. En este trabajo se utilizan las funciones

ReLU, sigmoide y *leaky ReLU*, que pasamos a describir a continuación.



Definición de la función de activación:

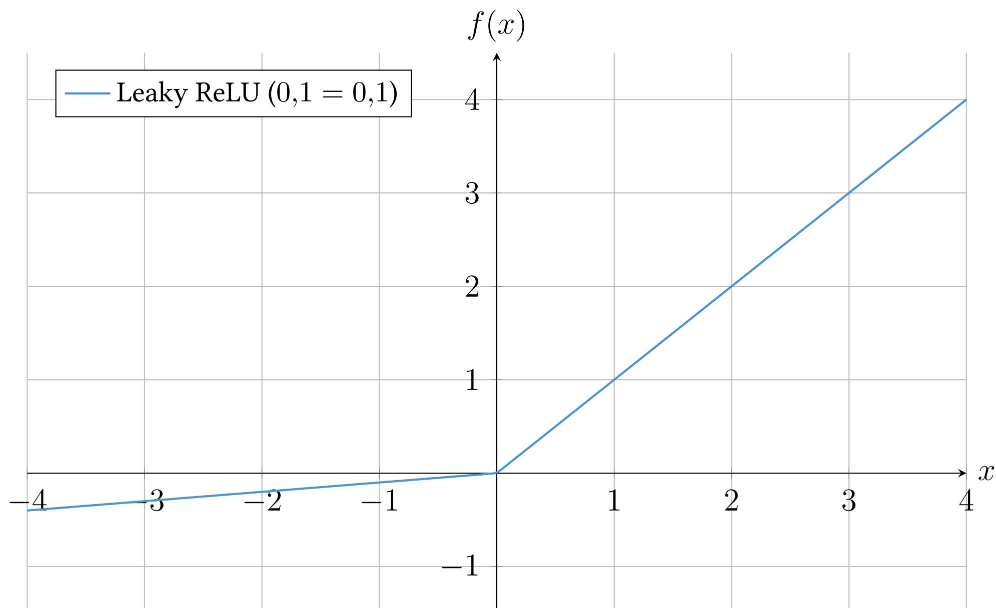
- **ReLU:** $\mathrm{ReLU}(x) = \max(x, 0)$ — es la función de activación más utilizada por su simplicidad y eficiencia computacional.

La función *ReLU* [11] actúa, en esencia, como un interruptor: para valores de entrada $x < 0$ apaga la neurona al devolver 0, y para $x > 0$ la enciende, dejando pasar la señal sin modificar. En la práctica, esto implica que cuando la salida es 0 las neuronas de las capas posteriores no intervienen, reduciendo de forma directa el número de operaciones aritméticas durante la inferencia. Al mismo tiempo, el gradiente de *ReLU* es constante e igual a 1 en el semieje positivo, lo que evita la saturación que sufren otras funciones de activación como la sigmoide o tangente hiperbólica y permite que los gradientes se propaguen sin atenuarse en redes profundas. En el caso de las redes neuronales especializadas en el tratamiento de imágenes, como las redes convolucionales la red debe aprender patrones en las imágenes, si una neurona detecta con claridad el patrón buscado, su potencial sináptico z es positivo y la señal se refuerza. Por otro lado, cuando el patrón no está presente, z es negativa, la salida se hace 0 y la información irrelevante se "descarta" de manera automática.

Sin embargo, la función *ReLU* también tiene inconvenientes. Uno de ellos es el *dying ReLU*, que ocurre cuando los pesos llevan la entrada de la neurona a valores siempre negativos, haciendo que su salida y su gradiente queden fijados en cero y la neurona deje de aportar

aprendizaje al modelo.

Para lidiar con este problema, se introduce la función *Leaky ReLU* [12].

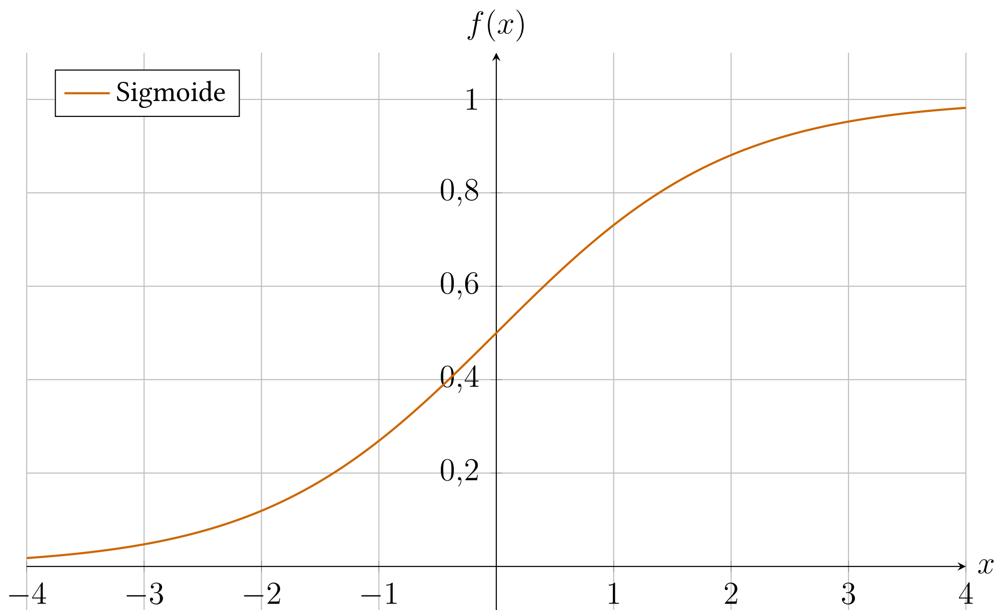


Definición de la función de activación:

- **Leaky ReLU:** $\mathrm{LeakyReLU}(x) = \begin{cases} \alpha\, x & \text{si } x < 0, \\ x & \text{si } x \geq 0, \end{cases}$

  donde α (habitualmente $0{,}01$) mantiene una pequeña pendiente en la zona negativa para evitar el problema de *dying ReLUs*. En este caso, $\alpha = 0,1$ a efectos ilustrativos.

Por último, definimos la función sigmoide.



Definición de la función de activación:

- **Sigmoide:** $\sigma(x) = \frac{1}{1+e^{-x}}$ — comprime la entrada al rango $[0, 1]$.

Produce salidas en el rango $[0, 1]$ y resulta útil en capas de salida para tareas de clasificación binaria. No obstante, satura en ambos extremos y presenta derivadas pequeñas, por lo que suele evitarse en capas ocultas profundas. En este proyecto, como se verá más adelante, esta función solo es usada en la capa final del modelo.

2.2. La neurona convolucional

En este trabajo se utiliza el tipo de neurona denominada neurona *convolucional*. Las operaciones de convolución son operaciones utilizadas para extraer características locales de una imagen, como bordes, texturas, o patrones visuales.

La operación de convolución. Una convolución consiste en aplicar un filtro o *kernel*[8] que se desliza por la imagen. En cada posición, se toma una submatriz de la imagen del mismo tamaño que el *kernel* y se realiza una multiplicación elemento a elemento entre ambas matrices. Los resultados de esa multiplicación se suman, se añade un término de sesgo o *bias* y el valor resultante se almacena en una nueva matriz llamada mapa de activación, que refleja la

[8]Kernel: Pequeña matriz de pesos de tamaño $m \times n$

presencia de ciertas características en diferentes regiones de la imagen. Cada valor del mapa representa el grado en que el patrón aprendido por el *kernel* (bordes, texturas, formas geométricas, etc.) se encuentra presente en una región específica de la imagen. Desde un punto de vista matemático, para una imagen I y un filtro K, ambos bidimensionales, la operación de convolución se define como:

$$S(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) \cdot K(m,n) + b \tag{2}$$

La operación de convolución visualmente se ve de la siguiente manera, representada en la figura 1.

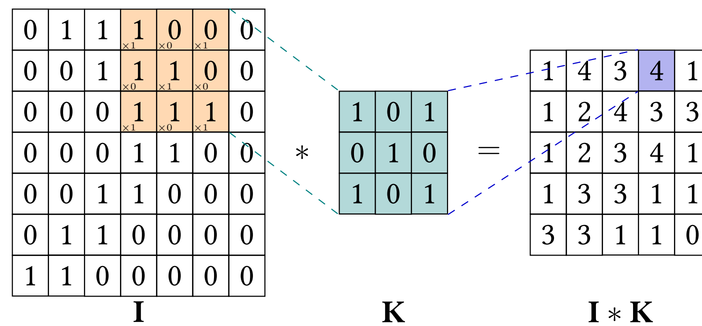


Figura 1: Ejemplo de convolución sobre imagen [13]

El movimiento de estos filtros o *kernels* se controlan con dos hiperparámetros, *stride* y *padding*, definidos a continuación. En la figura 2 se aprecia el deslizamiento del *kernel* sobre la imagen I.

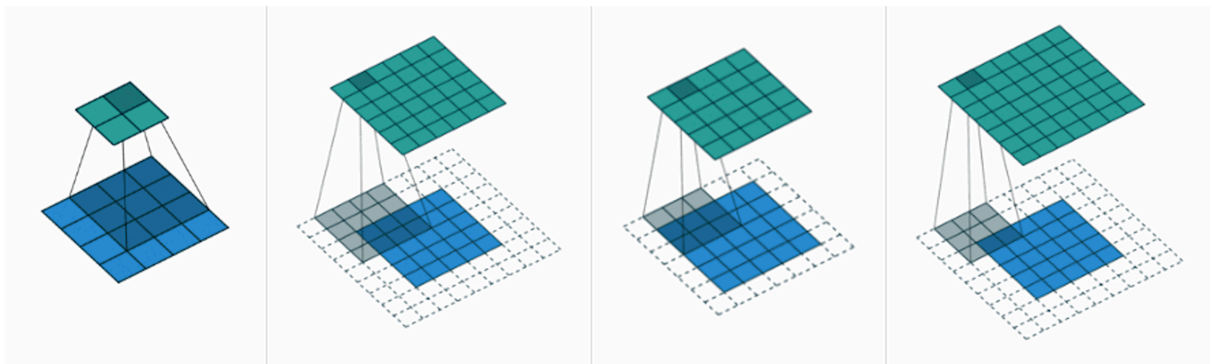


Figura 2: Deslizamiento del filtro sobre la imagen [14].

1. **Stride**: cuántos píxeles avanza el *kernel* en cada paso.

2. **Padding**: borde de ceros que se añade para mantener el tamaño o evitar perder información en los extremos.

Ideas básicas:

1. **Compartición de pesos**: Se utiliza el mismo *kernel* para recorrer toda la imagen. Al reutilizar el *kernel* en cada posición, la capa solo necesita aprender $K_H \times K_W \times C_{\text{in}}$ parámetros, donde C_{in} son los canales de entrada, en lugar de un parámetro distinto por píxel, lo que simplifica el modelo y reduce el riesgo de sobreajuste. Además, como el *kernel* responde del mismo modo sin importar dónde aparezca el patrón, la operación es traslacionalmente invariante, es decir, cualquier patrón se detecta con la misma fiabilidad tanto en la esquina como en el centro de la imagen.

2. **Filtros en paralelo**: Cada *kernel* actúa como un detector especializado y produce su propio mapa de activación. Si la entrada tiene C_{in} canales (por ejemplo, $C_{\text{in}} = 3$ para una imagen *RGB*) y aplicamos C_{out} *kernels*, obtendremos C_{out} mapas. Estos mapas se apilan formando un tensor de salida de tamaño (H', W', C_{out}), donde H' (*height*) y W' (*width*) representan la altura y la anchura del mapa de activación que surge tras la convolución. En otras palabras, la salida puede interpretarse como C_{out} imágenes de tamaño $H' \times W'$. Por ejemplo, una entrada de forma $(224, 224, 3)$ procesada con 64 filtros de tamaño 3×3 (sin *padding*) produce una salida $(222, 222, 64)$.

3. **Tamaño de salida**: Para una entrada de tamaño (H, W, C_{in}), un *kernel* de tamaño $K_h \times K_w$, *padding* P y *stride* S, la resolución de la salida viene dada por:

$$H' = \left\lfloor \frac{H + 2P - K_h}{S} \right\rfloor + 1, \qquad W' = \left\lfloor \frac{W + 2P - K_w}{S} \right\rfloor + 1. \tag{3}$$

2.3. De neuronas a redes

A continuación se enlazarán los conceptos de neurona y red. Como se ha mencionado anteriormente, una red neuronal está compuesta por capas interconectadas de neuronas. Las redes neuronales implementadas en este trabajo son de tipo *GAN* y *Autoencoder*. Además, se estudia la arquitectura con Transformers, en la que se basa el modelo *ICT* utilizado en este proyecto.

Estas tres arquitecturas trabajan con la misma base, las redes *feed-forward*. Una red *feed-forward* es el prototipo más simple de una red neuronal, y como su nombre indica, la información fluye siempre hacia delante, sin bucles ni retroalimentación.

La estructura general de este tipo de red es:

1. **Entrada**: Un vector $x \in \mathbb{R}^{d_0}$, donde d_0 indica el número de características de entrada, que contiene las características de un ejemplo (o una matriz $X \in \mathbb{R}^{N \times d_0}$ para un lote de N ejemplos).

2. **Capas ocultas**: Procesan la información y aprenden patrones complejos en los datos.

3. **Salida**: Un vector $y \in \mathbb{R}^{d_L}$ producido por la última capa, cuyo tamaño d_L se elige según la tarea.

Cada capa oculta *i* aplica la transformación $\mathbf{h_i} = \sigma_\mathbf{i}\big(\mathbf{W_i}\,\mathbf{h_{i-1}} + \mathbf{b_i}\big)$, donde W es la matriz de pesos, h_{i-1} es la entrada, y b el sesgo. A este cálculo se le aplica la función de activación σ, que dependiendo de la tarea a realizar, puede ser, por ejemplo, una función sigmoide para tareas de clasificación binaria, debido a su salida $[0, 1]$.

Cuando la red tiene varias capas completamente conectadas, es decir, todas las neuronas de la capa anterior se conectan a todas las neuronas de la siguiente capa, se la llama *MLP (Multi-layer perceptron)*. En contraste, en una capa convolucional cada neurona se conecta únicamente a un campo receptivo local de la capa anterior, es decir, la región de entrada definida por el filtro, y dicho filtro se aplica con los mismos pesos en todas las ubicaciones.

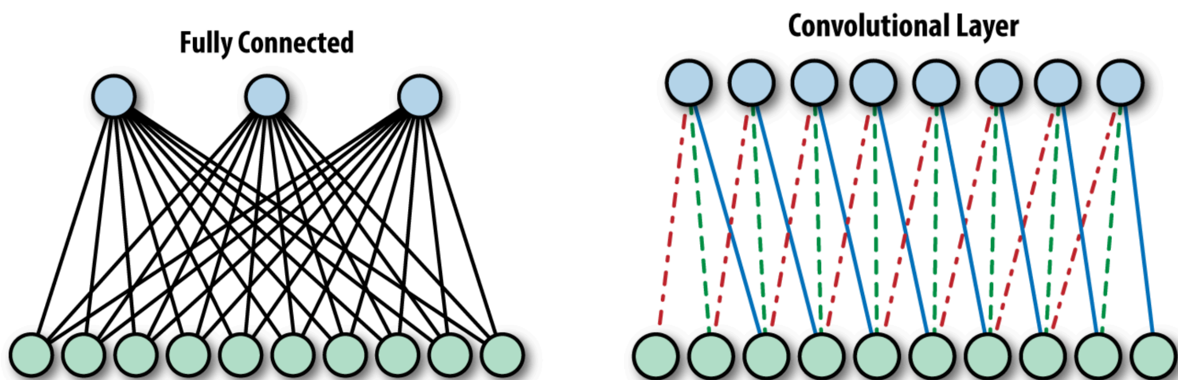


Figura 3: Diferencia visual entre una capa MLP y una capa convolucional [15].

La capacidad de una red para aprender patrones complejos depende de varios factores, como su profundidad o número de capas, el número de neuronas por capa y del número total de conexiones entre capas. Cuantas más capas y más neuronas por capa tenga, y por tanto más conexiones, podrá descubrir patrones más complejos, por ejemplo, de bordes a texturas y de texturas a objetos completos.

Los datos se pueden presentar a la red de forma individual o en lotes, también llamados *batches*. Tanto en uno como en el otro caso, los datos se organizan en estructuras de datos llamadas tensores, que tienen una dimensión fija que permiten a la red procesar varios ejemplos simultáneamente y así aprovechar que las GPU están preparadas para procesar varios bloques de datos a la vez.

Un lote o *batch* de N imágenes se representa como $\mathbf{X} \in \mathbb{R}^{N \times H \times W \times C_{\text{in}}}$, con C_{in} los canales de color (RGB), y H y W la altura y anchura respectivamente. De esa manera, cada capa puede aplicar la misma transformación a todas las muestras simultáneamente, manteniendo así la eficiencia.

2.4. Entrenamiento de redes neuronales

Una red neuronal aprende realizando predicciones sobre un conjunto de datos. Cada predicción se compara con el valor real o esperado mediante una función llamada función de pérdida, que devuelve el error cometido por la red neuronal, es decir, la distancia entre la predicción y lo que realmente debería haber predicho.

A partir de esta pérdida, la red optimiza sus pesos, modificándolos ligeramente en la dirección que reduzca dicha pérdida. Este proceso se repite con el objetivo de minimizar la función de pérdida, logrando así que la red neuronal haga predicciones cada vez más precisas.

Una función de pérdida típica para tareas de regresión es el error cuadrático medio (*MSE*) o para tareas de clasificación binaria, la entropía cruzada binaria (*BCE*):

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}_i - y_i\right)^2 \tag{4}$$

donde $\hat{y}_i$ es la predicción de la red y y_i su valor esperado o real.

$$\text{BCE} = -\frac{1}{N} \sum_{i=1}^{N} \Big[t_i \log\left(p_i\right) + (1 - t_i) \log\left(1 - p_i\right) \Big] \tag{5}$$

donde p_i es la probabilidad predicha y $t_i \in \{0, 1\}$ la etiqueta verdadera.

La red ajusta sus pesos a través de un algoritmo llamado *backpropagation*. Este algoritmo es un proceso que calcula el gradiente de la función de pérdida con respecto a cada peso en la red. Este gradiente se usa para actualizar los pesos en la dirección contraria a la que marca el gradiente, y esto a su vez minimiza el error al buscar un mínimo en la función de pérdida.

El entrenamiento se realiza en dos pasos [16] [17]:

1. ***Forward propagation***: Los datos de entrada pasan por la red y los pesos se multiplican con las entradas para calcular la salida, que será:

$$\mathbf{z}^{(l)} = W^{(l)}\,\mathbf{h}^{(l-1)} + \mathbf{b}^{(l)}, \quad \mathbf{h}^{(l)} = \sigma\big(\mathbf{z}^{(l)}\big), \quad l = 1, \ldots, L, \tag{6}$$

   con $\mathbf{h}^{(0)} = \mathbf{x}$, L el número de capas y salida final $\mathbf{h}^{(L)}$.

   Tras ello, se calcula la pérdida entre el dato real y la salida de la red con la función de pérdida seleccionada.

2. ***Backpropagation***: La pérdida calculada en el paso anterior se propaga a través de la red para calcular el gradiente de la función de pérdida. Esta propagación se realiza usando la regla de la cadena y consiste en estimar cuánto contribuye cada peso a la salida final, siguiendo los siguientes pasos.
   Definimos el error local en la capa l como

$$\boldsymbol{\delta}^{(l)} \;=\; \frac{\partial E}{\partial \mathbf{z}^{(l)}} \;=\; \frac{\partial E}{\partial \mathbf{h}^{(l)}} \;\circ\; \sigma'\big(\mathbf{z}^{(l)}\big). \tag{7}$$

   Dado que el potencial sináptico $\mathbf{h}$ depende de los pesos sinápticos $\mathbf{W}$, los gradientes de la función de pérdida E respecto a los pesos son

$$\frac{\partial E}{\partial W^{(l)}} \tag{8}$$

   Finalmente, cada peso y *bias* se actualiza con descenso del gradiente:

$$W^{(l)} \leftarrow W^{(l)} - \eta\,\frac{\partial E}{\partial W^{(l)}}, \quad \mathbf{b}^{(l)} \leftarrow \mathbf{b}^{(l)} - \eta\,\frac{\partial E}{\partial \mathbf{b}^{(l)}} \tag{9}$$

   donde η es el *learning rate*.

Los pesos se actualizan restando una fracción del gradiente calculado, que se determina con el *learning rate*, o tasa de aprendizaje, que define cuánto pueden cambiar los pesos en cada iteración.

Al cabo de varias iteraciones, este proceso de *forward y backward propagation* converge cuando los gradientes se aproximan a cero, es decir, cuando las actualizaciones de los pesos se hacen tan pequeñas que prácticamente no modifican la función de pérdida. En ese punto la red está entrenada, se ha encontrado un mínimo de la función de pérdida y sus parámetros se han optimizado según los datos de entrenamiento.

En la práctica, estas actualizaciones de los pesos se implementan a través de un optimizador, que es el componente encargado de calcular y aplicar las modificaciones a los pesos en cada iteración. Un ejemplo muy común es el optimizador *Adam (Adaptive Moment Estimation)*, que combina el momento con el optimizador *RMSprop (Root Mean Square Propagation)*. *Adam* se define como [18]:

1. **Momento (primer momento):** Mantiene una media móvil de los gradientes pasados para acelerar la convergencia:

$$m_t = \beta_1 \, m_{t-1} + (1 - \beta_1) \, \frac{\partial E}{\partial W^{(l)}}, \tag{10}$$

donde m_t es la media móvil del gradiente en el paso t, $\beta_1 \in [0, 1)$ es el coeficiente de decaimiento, y $\frac{\partial \mathcal{E}}{\partial W^{(l)}}$ el gradiente actual.

2. ***RMSprop* (segundo momento):** Ajusta la tasa de aprendizaje de cada parámetro según la media móvil del cuadrado del gradiente:

$$v_t = \beta_2 \, v_{t-1} + (1 - \beta_2) \left(\tfrac{\partial \mathcal{L}}{\partial \omega_t}\right)^2, \qquad \omega_{t+1} = \omega_t - \frac{\alpha}{\sqrt{v_t} + \varepsilon} \, \frac{\partial \mathcal{L}}{\partial \omega_t}, \tag{11}$$

donde v_t es la media móvil de los gradientes al cuadrado, α la tasa de aprendizaje y ε un término de estabilidad.

3. ***Adam* (combinación de ambos):**

   a) Estimación del primer momento (momento):

$$m_t = \beta_1 \, m_{t-1} + (1 - \beta_1) \, \frac{\partial \mathcal{L}}{\partial \omega_t} \tag{12}$$

b) Estimación del segundo momento (*RMSprop*):

$$v_t = \beta_2 \, v_{t-1} + (1 - \beta_2) \left(\frac{\partial \mathcal{L}}{\partial \omega_t} \right)^2 \tag{13}$$

c) Corrección de sesgo:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{14}$$

d) Actualización de parámetros:

$$\omega_{t+1} = \omega_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \, \hat{m}_t. \tag{15}$$

En *Adam* se definen los siguientes valores por defecto: $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$, $\varepsilon = 10^{-8}$ y $\alpha = 10^{-3}$ [19], los cuales permiten combinar el impulso del momento con la adaptación de *RMSprop* para un entrenamiento rápido y estable.

No obstante, que el gradiente de la función de pérdida sea cercano a cero en el conjunto de entrenamiento no garantiza que la red haya sido entrenada correctamente. Existe un riesgo de *overfitting* o sobreajuste, que ocurre cuando la red ha memorizado los ejemplos de entrenamiento. Esto le hace perder generalidad, lo que significa que al entrar datos nuevos en la red que no aparecían en el conjunto de datos de entrenamiento, la red puede procesarlos incorrectamente.

Para evitarlo, se recurre a técnicas como el *early stopping*, que detiene el entrenamiento cuando la pérdida deja de mejorar o *data augmentation*, que es un proceso de aumento del conjunto de datos de entrenamiento, evitando que la red entrene repetidamente sobre los mismos datos y reduciendo así el riesgo de "memorizar" los datos de entrenamiento.

2.5. Tipos de capas en una red

Dentro de una red se combinan distintos tipos de capas, cada una de las cuales tiene una función distinta. Las capas más comunes son:

1. **Capa convolucional**: Las neuronas que componen estas capas son convolucionales. Como anteriormente se ha definido, la operación de convolución desliza un *kernel* por la imagen de entrada, convirtiéndola en mapas de activación que destacan las características que busca el *kernel*.

2. **Capa de *pooling***: Reduce la dimensión de la entrada, reteniendo la información más importante y descartando el resto. En este trabajo se ha usado *max-pooling*, que dado un tamaño y un *stride*, devuelve el máximo valor, tal y como se ve en la figura 4.

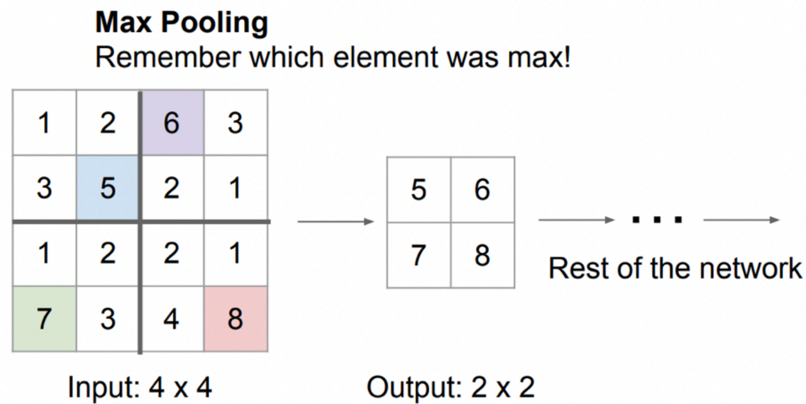


Figura 4: Capa de *max-pooling* [20].

3. **Capa de normalización**: Normaliza los datos de entrada hacia una distribución centrada en 0 y una desviación estándar 1.

4. **Capa de upsampling**: Aumenta la dimensión de la entrada, en este trabajo se usa el tipo de interpolación *nearest neighbour*, que para cada nuevo píxel asigna el valor del píxel existente más cercano en la imagen original, tal y como se ve en la figura 5

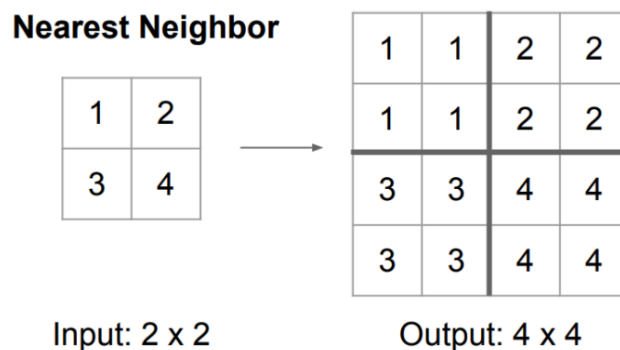


Figura 5: Nearest neighbour [20].

5. **Capa densa o *fully-connected***: Las capas densas, o totalmente conectadas, son aquellas en las que cada neurona recibe como entrada la señal de todas las neuronas de la capa anterior. Son el tipo de capa más común, especialmente en redes neuronales destinadas a tareas de clasificación.

2.6. Arquitecturas profundas especializadas

En este apartado se detallará el funcionamiento de las tres arquitecturas usadas en el trabajo: *Autoencoders*, *GANs* y *Transformers*.

2.7. Autoencoders

Los *Autoencoders* son conocidos por su capacidad para aprender representaciones comprimidas de imágenes y generar reconstrucciones coherentes incluso a partir de información parcial. El *Autoencoder* implementado en este trabajo es llamado un *Autoencoder* convolucional, lo que significa que cada neurona, tanto en la parte codificadora como decodificadora, aplica una operación de convolución sobre su "ventana de trabajo". En la primera capa, esa ventana corresponde a un parche de la imagen original, y en capas más profundas, al parche correspondiente de un mapa de activación anterior. Un *Autoencoder* consta de dos bloques esenciales: el *encoder* y el *decoder*.

En el *encoder* se aplican sucesivamente capas de convolución y activaciones no lineales como *ReLU*. Cada paso de convolución va seguido de una operación de *pooling*, con la cual se reduce el tamaño de la representación de la imagen, hasta llegar al *bottleneck* o cuello de botella[9]. En ese punto la representación de los datos de entradas se conoce como vector latente, que obliga a la red a conservar solo las características más relevantes.

Posteriormente, el *decoder* se ocupará de la tarea de reconstrucción, aplicando en orden una capa de upsampling con interpolación *nearest-neighbor*, con las cuales se expande la representación latente, seguida de una capa de convolución. Este ciclo se repite hasta hasta finalmente volver a las dimensiones de la imagen original.

Este diseño *encoder–bottleneck–decoder* resulta útil para tareas de reconstrucción, ya que el cuello de botella obliga a la red a capturar el contexto global y el *decoder* puede rellenar

[9]Cuello de botella: representación de la imagen más comprimida

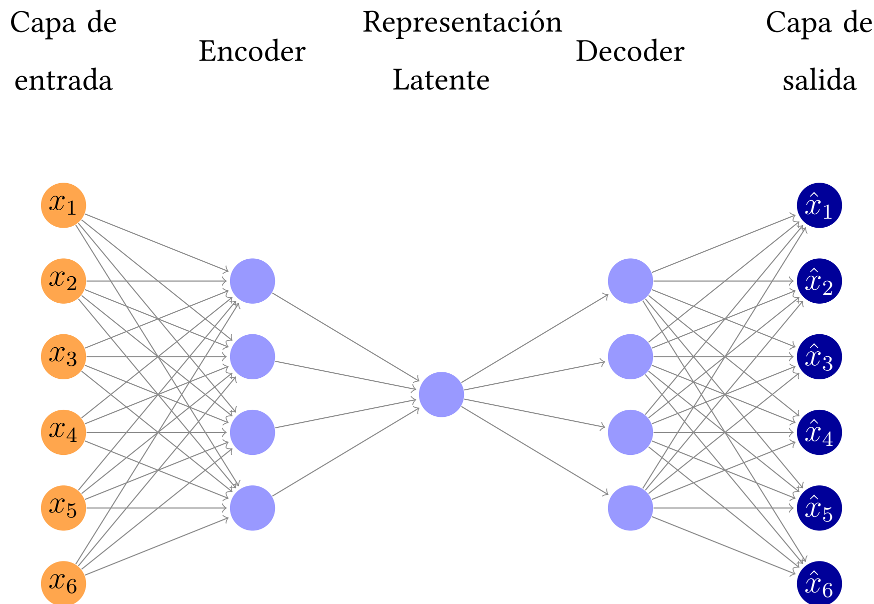


Figura 6: Representación de un *Autoencoder* [13]

huecos y reparar daños de forma coherente con el estilo de la imagen original.

Dentro de los *Autoencoders* existen distintos tipos de arquitecturas. Una de ellas que será usada en el desarrollo del *Autoencoder* en este trabajo es la arquitectura *U-Net* [21]. Es especialmente útil en tareas de reconstrucción, ya que ayuda a preservar bordes, contornos y otras estructuras locales que de otro modo podrían perderse al pasar por el cuello de botella. Esto es gracias a los *skip connections*, que son conexiones que enlazan directamente cada capa del *encoder* con su correspondiente capa del *decoder*. De esta forma, el *decoder* no depende únicamente de la información del cuello de botella, sino que también recupera detalles de etapas anteriores, como bordes, formas o texturas. Un *skip connection* tiene la forma que se muestra en la figura 7.

En la arquitectura *U-Net* se guardan los mapas de activación de cada capa de convolución del *encoder* antes de la operación de *pooling*, para así posteriormente concatenarlas con cada salida de upsampling del *decoder*. De esta manera la entrada a cada capa de convolución del *decoder* recibe como entrada un tensor con los detalles locales originales, pudiendo así realizar la reconstrucción sin tener que basarse únicamente en la representación latente del cuello de botella.

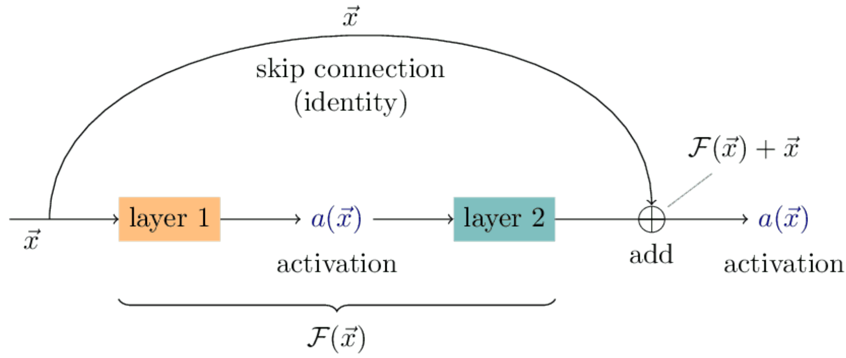


Figura 7: Diagrama de skip connection [13]

2.8. GANs

Las redes adversarias generativas, o *GANs*, son un tipo de modelo de aprendizaje profundo en el que dos redes, un discriminador y un generador, se enfrentan en un juego de suma cero. La tarea del discriminador es distinguir imágenes verdaderas, procedentes de la distribución de datos $p_{\text{data}}(x)$, de imágenes artificialmente creadas. La tarea del generador es crear imágenes lo suficientemente realistas, a partir de un ruido $z \sim p_z(z)$, como para confundir al discriminador de su procedencia.

Este enfrentamiento o competición entre ambas redes se formaliza mediante la función *Min-Max*, que se define como:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} \big[\log D(x)\big] + \mathbb{E}_{z \sim p_z(z)} \big[\log\big(1 - D(G(z))\big)\big].$$

Donde el generador G minimiza la función y el discriminador D la maximiza. Aquí $D(x) \in [0, 1]$, que es la salida del discriminador, representa la probabilidad de que x sea real, donde 0 es certeza absoluta de que la imagen x es falsa y 1 certeza absoluta de que es verdadera. $G(z)$, por otro lado, se refiere a la salida del generador.

Discriminador El discriminador debe aprender a distinguir de forma precisa entre imágenes auténticas y generadas. En la función *Min-Max* se definen los términos $\log D(x)$, que corresponde a la probabilidad logarítmica de que una imagen real sea correctamente identificada como tal, y $\log(1 - D(G(z)))$, que representa la probabilidad logarítmica de clasificar como falsa una imagen producida por el generador.

Visto de otra manera, si la imagen es real:

1. Si el discriminador adivina correctamente que es una imagen real, la salida será cercana a 1 y el término $\log D(x)$ alcanza su valor máximo (igual a cero) si $D(x) = 1$.
2. Si el discriminador adivina incorrectamente que es falsa, la salida será negativa, ya que cuando $D(x) < 1$, el logaritmo será negativo (por ejemplo, $\log(0{,}2) \approx -0{,}69$), penalizando la salida equivocada y empujando los pesos para que la próxima vez la predicción se acerque más a 1.

Cuando la imagen es falsa interviene el generador, por lo que ahora el segundo término $\log(1 - D(G(z)))$ determina la pérdida:

1. Si el discriminador adivina que la imagen falsa es falsa, la salida será cercana a 0, ya que adivina que $D(G(z)) \approx 0$ y $\log(1 - 0) = 0$.
2. Si el discriminador se equivoca y ofrece una probabilidad positiva para la imagen falsa, el logaritmo $\log(1 - D(G(z)))$ se vuelve negativo, lo que de nuevo causa una corrección en los pesos.

En conjunto, ambos términos comparten el cero como valor óptimo y cualquier predicción incorrecta produce valores negativos en el valor total de la función *Min-Max*. Por esta razón el discriminador afronta un problema de maximización, su objetivo es llevar cada uno de los logaritmos lo más cerca posible de su cota superior, que en el caso de la función logarítmica es cero, ajustando los pesos de manera que las imágenes reales reciban salida $D(x) = 1$ y las falsas $D(G(z)) = 0$.

Generador El generador tiene dos objetivos, producir imágenes suficientemente realistas y engañar al discriminador para que las clasifique como auténticas. Estos dos propósitos requieren un equilibrio, ya que no se debe permitir que el generador priorice una tarea antes que otra.

En el artículo original *Generative Adversarial Networks* [22], donde se presenta por primera vez la arquitectura *GAN*, se propone reemplazar el término $\log(1 - D(G(z)))$ por la forma $-log(D(G(z)))$ en la función de pérdida del generador. Esto se debe a que al utilizar $\log(1 - D(G(z)))$, los gradientes que recibe el generador se vuelven muy pequeños cuando $D(G(z))$ se aproxima a cero, impidiendo que el generador aprenda de manera efectiva. Al sustituirlo por

$-log(D(G(z)))$ los gradientes mantienen una magnitud razonable durante toda la duración del entrenamiento, por lo tanto se usará la segunda formulación del término.

Los dos casos que se dan en el generador son:

1. El generador presenta una imagen y el discriminador la categoriza como falsa. El generador pierde el juego y se refleja en el término como que $D(G(z)) \approx 0$ (por ejemplo $-\log(0{,}1) \approx 2{,}3$, una pérdida de $+2{,}3$ muy grande para el generador.

2. El generador presenta una imagen y el discriminador la categoriza como verdadera. El generador gana el juego y en el término se refleja como $D(G(z)) \approx 1$. Por ejemplo $-log(0{,}9) \approx 0{,}1$, una pérdida baja para el generador.

Se aprecia pues, como para un caso el discriminador intenta maximizar la función objetivo, ya que el rango de sus valores dependen de los términos logarítmicos, que toman valores en $[-\infty, 0]$, de modo que su cota superior es 0.

Para el generador se intentará minimizar la función ya que el signo negativo invierte la función logarítmica, de forma que cuanto más cercano a cero sea $D(G(z))$, mayor será la pérdida y más fuerte la corrección de los pesos.

Se considera que un modelo *GAN* ha convergido cuando la precisión o *accuracy* del discriminador se estabiliza alrededor de $0{,}5$, valor que indica que ya no distingue entre imágenes reales y sintéticas y que su decisión es comparable a lanzar una moneda al aire. Sin embargo, llegar a una precisión de $0{,}5$ no significa que el modelo está correctamente entrenado, existe un fenómeno llamado *mode collapse*, que ocurre cuando el generador encuentra un subconjunto de imágenes que engañan al discriminador y sigue generando esas mismas imágenes continuamente para diferentes entradas. En este trabajo se han monitorizado los parámetros de pérdida del discriminador y generador durante el entrenamiento para asegurar que no ocurre este suceso.

2.9. Transformers

Los *Transformers* emplean una arquitectura basada en aprendizaje profundo que ha demostrado ser especialmente efectiva en tareas de procesamiento de lenguaje natural. Sus componentes y mecanismos clave son los siguientes:

Mecanismo de atención Los *Transformers* revolucionaron el mecanismo de atención. Citando al artículo original, *Attention is all you need* [23], el *Transformer* es el primer modelo de transducción que se basa completamente en la auto-atención o *self-attention* para calcular representaciones de su entrada y salida sin emplear redes neuronales recurrentes ni convoluciones.

Este nuevo mecanismo se basa en los siguientes componentes [24]:

1. **q** y **k**: ambos son vectores de dimensión d_k, contienen las consultas y claves.

2. **v**: es un vector de dimensión d_v, contiene los valores.

3. **Q**, **K** y **V**: cada uno de estos componentes se refiere, respectivamente, a: *query*, que es la representación vectorial de una palabra en la frase; *key*, que actúa como un índice (análogo al índice de un libro) que se compara con **Q** para responder a la pregunta "¿contiene la palabra procesada X la información que yo (**Q**) estoy buscando?"; y, por último *value* **V**, que agrupa y filtra la información correspondiente al contenido de las páginas que señala el índice **K** para generar la salida final. Matemáticamente se definen como:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}_1 \\ \mathbf{q}_2 \\ \vdots \\ \mathbf{q}_n \end{bmatrix} \in \mathbb{R}^{n \times d_k}, \quad \mathbf{K} = \begin{bmatrix} \mathbf{k}_1 \\ \mathbf{k}_2 \\ \vdots \\ \mathbf{k}_n \end{bmatrix} \in \mathbb{R}^{n \times d_k}, \quad \mathbf{V} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix} \in \mathbb{R}^{n \times d_v} \tag{16}$$

4. $\mathbf{W}^Q$, $\mathbf{W}^K$ y $\mathbf{W}^V$: son las matrices de proyección que se utilizan para generar diferentes representaciones de subespacios de las matrices de query, clave y valor.

5. $\mathbf{W}^O$: es la matriz de proyección para la salida del multi-head attention, que definiremos más adelante.

Estos valores son la entrada del mecanismo multi-head attention, que se representa como indica la Figura 8:

La figura de la derecha muestra como la arquitectura multi-head es una paralelización del *scaled dot-product attention*. La salida del *scaled dot-product attention* se define como:

$$\text{attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\,\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \tag{17}$$

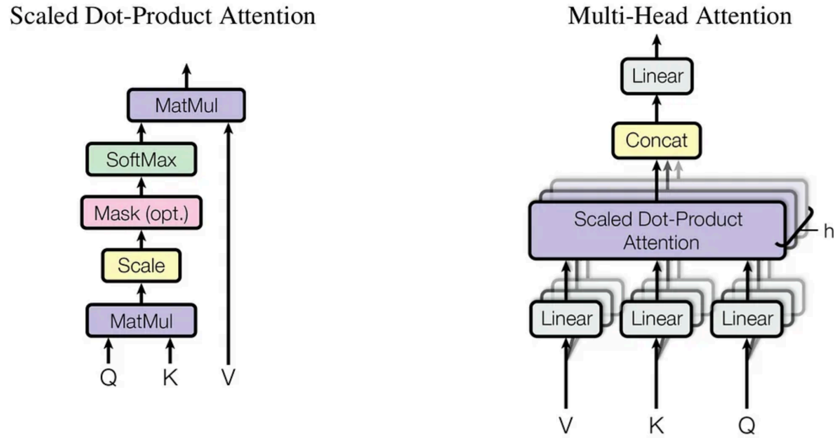


Figura 8: Diagrama del mecanismo multi-head- attention[25]

El mecanismo de *multi-head attention* proyecta linealmente las consultas, claves y valores h veces, como se ve en la figura 8. El mecanismo *scaled dot-product attention* se aplica a cada una de estas proyecciones en paralelo para producir h resultados, que, a su vez, se concatenan y se proyectan de nuevo para producir un resultado final. Esta concatenación se define como:

$$\text{multihead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) \;=\; \text{concat}\big(\text{head}_1, \ldots, \text{head}_h\big)\, \mathbf{W}^O$$

Donde cada $head_i$ es:

$$\text{head}_i \;=\; \text{attention}\big(\mathbf{Q}\,\mathbf{W}_i^Q,\; \mathbf{K}\,\mathbf{W}_i^K,\; \mathbf{V}\,\mathbf{W}_i^V\big)$$

Encoder y decoder Al igual que un *Autoencoder*, el *Transformer* es una arquitectura para transformar una secuencia en otra con la ayuda de dos partes, un *encoder* y *decoder*.

En la figura 9, el *encoder* está a la izquierda y el *decoder* a la derecha. Tanto el *encoder* como el *decoder* están compuestos por módulos que se pueden apilar uno encima del otro N veces. Se puede apreciar como los módulos consisten principalmente en capas *multi-head attention* y *feed forward*.

Paso del *Transformer* clásico al *ViT (Visual Transformer)* Para aplicar la implementación del *Transformer* clásico a imágenes, se presenta en el artículo *An Image is Worth 16x16 Words* [26] una modificación llamada *image tokenization*, donde la imagen se divide en parches

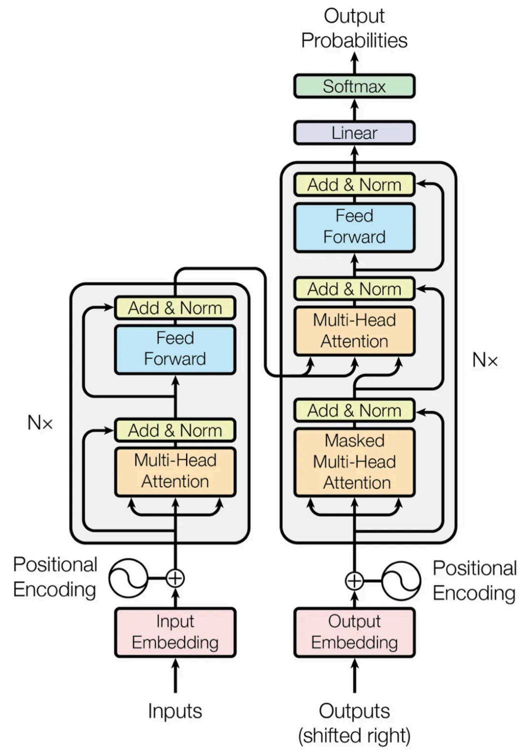


Figura 9: Estructura de un *Transformer*[25]

y cada parche está representado por un *token*.

El número de *tokens* que representa una imagen viene dado por $N = \frac{HW}{P^2}$, donde H es la altura, W el ancho de la imagen y P el tamaño del parche. Cada *token* tendrá un tamaño de $P^2 \times C$, donde C son los canales de la imagen, tal y como se ve en la figura 11.

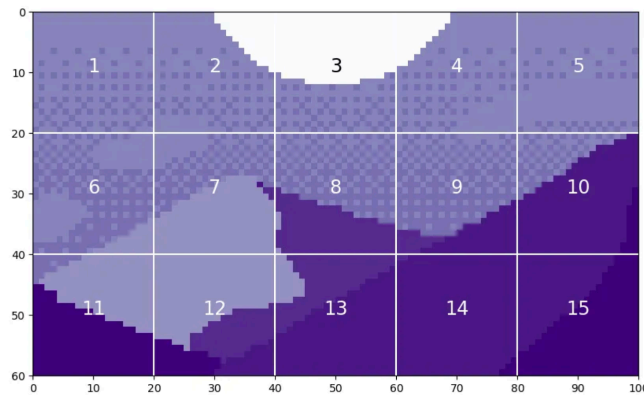


Figura 10: División de imagen en *tokens* [27].

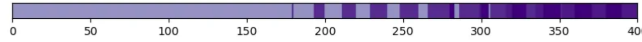


Figura 11: Ejemplo de *token* extraído de la imagen [27].

A continuación se antepone un *token* llamado *prediction token*, o *token* de predicción, que inicialmente estará inicializado a cero y servirá para recolectar información de los demás *tokens* a lo largo de los bloques de codificación.

Por último, antes de entrar al *encoder*, se suma un *embedding* posicional, que codifica su posición relativa en la imagen. Este proceso se representa en la figura 12.

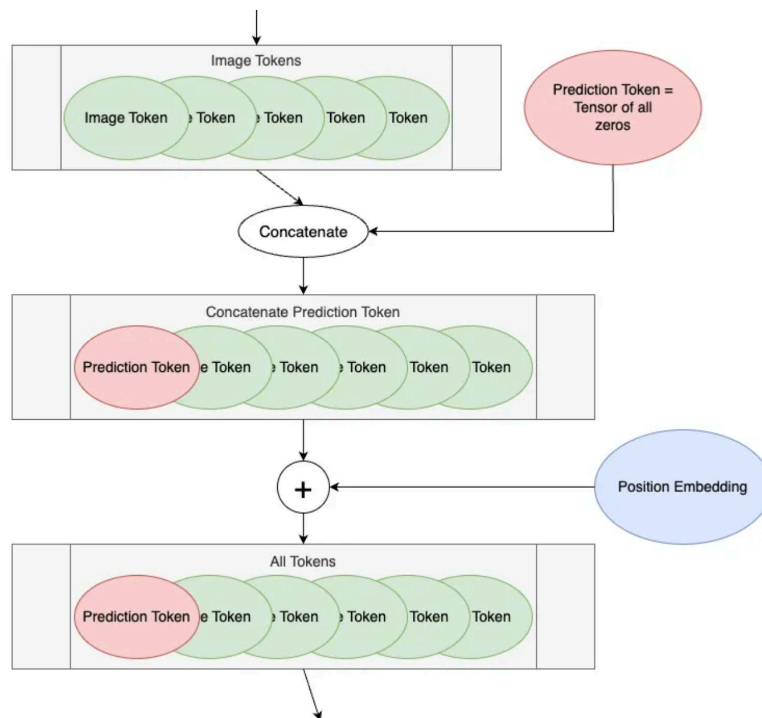


Figura 12: Preprocesamiento de los *tokens* [27].

Una vez los *tokens* entran en el *encoder*, el proceso es análogo al procesamiento de lenguaje natural descrito anteriormente.

2.10. Medidas de entrenamiento y rendimiento

Durante el entrenamiento de todos los modelos, las métricas *MSE*, *L1* y *SSIM* fueron empleadas como funciones de pérdida objetivo que el optimizador minimizó en cada paso. En cambio, el *accuracy* se imprimía junto con la curva de loss únicamente para monitorizar el

rendimiento del discriminador de la *GAN*. A continuación, se detallan estas métricas:

1. ***Accuracy***: Proporción de predicciones correctas sobre el total de predicciones realizadas. Se utiliza principalmente en tareas de clasificación.

   Como se ha visto anteriormente, el discriminador de un modelo *GAN* desempeña el papel de un clasificador binario (imagen real/falsa), por lo que su rendimiento se mide con esta métrica [28] .

$$\text{accuracy}(y, \hat{y}) \;=\; \frac{1}{N} \sum_{i=0}^{N-1} \mathbf{1}\big(\,\hat{y}_i = y_i\big) \tag{18}$$

   donde N es el número de muestras, y_i es la etiqueta real de la muestra i, $\hat{y}_i$ es la predicción y $\mathbf{1}$ es la función indicadora, definida como:

$$\mathbf{1}\big(\hat{y}_i = y_i\big) \;=\; \begin{cases} 1, & \text{si } \hat{y}_i = y_i, \\ 0, & \text{si } \hat{y}_i \neq y_i. \end{cases} \tag{19}$$

2. **Mean Squared Error (*MSE*)**: Error cuadrático medio entre valores predichos y valores reales. Es común en regresión y tareas de reconstrucción de imágenes.

$$\text{MSE} \;=\; \frac{1}{N} \sum_{i=1}^{N} \big(\hat{y}_i - y_i\big)^2 \tag{20}$$

   donde y_i es el valor real (o el píxel real) de la muestra i y $\hat{y}_i$ es el valor predicho correspondiente.

3. **Structural Similarity Index (*SSIM*)**: Índice que mide la similitud estructural entre dos imágenes x e y. La formulación original de *SSIM* es [29]:

$$\text{SSIM}(x, y) \;=\; \frac{\big(2\,\mu_x\,\mu_y + C_1\big)\,\big(2\,\sigma_{xy} + C_2\big)}{\big(\mu_x^2 + \mu_y^2 + C_1\big)\,\big(\sigma_x^2 + \sigma_y^2 + C_2\big)} \tag{21}$$

   En este trabajo se usa la implementación de *SSIM* de Tensorflow [30].

4. **Pérdida *L1***: También llamada error absoluto medio (MAE, por sus siglas en inglés), mide la diferencia absoluta entre predicciones $\hat{y}_i$ y valores reales y_i. Su fórmula es

$$\mathcal{L}_1 \;=\; \frac{1}{N} \sum_{i=1}^{N} \big|\hat{y}_i - y_i\big| \tag{22}$$

En tareas de reconstrucción de imágenes, es preferible usar la pérdida *L1* a *MSE*, ya que penaliza proporcionalmente cada discrepancia de píxeles, a diferencia de *MSE*, que magnifica los valores atípicos, de modo que el gradiente tiende a "repartir" ese ajuste entre varios píxeles, lo que a menudo provoca bordes menos nítidos.

Tras completar el entrenamiento de un modelo de aprendizaje automático, es esencial contar con métricas que ayuden a evaluar su desempeño y ver hasta qué punto puede generalizar a datos nuevos. Para ello, contamos con diferentes medidas de rendimiento que cuantifican numéricamente el rendimiento del modelo. En esta sección se definirán algunas de las métricas más relevantes en la evaluación del rendimiento de modelos como los *GANs* y *Transformers*, que usaremos más adelante:

1. ***FSIM (Feature Similarity Index)*** **[31]**: Mide la similitud estructural basándose en la magnitud de gradiente local y detectando bordes y puntos de inflexión. Es muy sensible a la preservación de contornos y estructuras finas, lo cual es clave al evaluar la fidelidad de los bordes y la topología global de la obra.

2. ***DISTS (Deep Image Structure and Texture Similarity)*** **[32]**: Combina características de alto nivel, como estructuras, y bajo nivel, como texturas, extraídas de una red neuronal entrenada para distinguir texturas. Equilibra la comparación de patrones globales y detalles locales.

3. ***LPIPS (Learned Perceptual Image Patch Similarity)*** **[33]**: Calcula la distancia perceptual entre parches de imagen usando activaciones intermedias de una CNN preentrenada. Mide errores que saltan a la vista en texturas y colores.

4. ***ORB (Oriented FAST and Rotated BRIEF)*** **[34]**: Detecta puntos de interés y los empareja entre la imagen original y la reconstrucción. Evalúa la preservación de patrones locales y la coherencia geométrica básica.

2.11. Contraste de hipótesis

Una vez calculadas las métricas definidas anteriormente, se realizará un análisis cuantitativo que complementará los resultados cualitativos de las imágenes reconstruidas, permitiendo evaluar de forma objetiva el rendimiento de los modelos.

Para el análisis, en primer lugar se define la diferencia de los errores para cada imagen i como d_i:

$$d_i = \begin{cases} \text{GAN} - \text{ICT}, & \text{para LPIPS, DISTS y MSE (menor es mejor).} \\ \text{ICT} - \text{GAN}, & \text{para SSIM, FSIM y ORB (mayor es mejor).} \end{cases} \tag{23}$$

Para las pruebas que se definirán a continuación, se utilizará la diferencia media, es decir:

$$\Delta = \frac{1}{N}\sum_{i=1}^{N} d_i, \text{donde } N \text{ es el número de imágenes.} \tag{24}$$

Donde $\Delta > 0$ indica que *ICT* mejora al *GAN* y $\Delta < 0$ que *GAN* supera al *ICT* para esa métrica concreta.

Seguidamente, se aplicará el test de *Shapiro-Wilk*[35] sobre los valores de Δ para comprobar la hipótesis nula H_0 frente a la alternativa H_1:

$$H_0 : \text{los datos provienen de una distribución normal.}$$

$$H_1 : \text{los datos no provienen de una distribución normal.}$$

Se obtiene el valor p y se rechaza la hipótesis nula cuando $p \leq 0{,}05$.

Cuando se rechaza la normalidad se emplearía el test de *Wilcoxon*. En este caso, las hipótesis enfrentadas serán:

$$H_0 : \text{el modelo } \textit{ICT} \text{ y } \textit{GAN} \text{ son equivalentes en rendimiento.}$$

$$H_1 : \text{el modelo } \textit{ICT} \text{ y } \textit{GAN} \text{ no son equivalentes en rendimiento.}$$

El test de *Wilcoxon* devolverá el valor p asociado, de nuevo, se rechaza la hipótesis nula cuando $p \leq 0{,}05$.

A partir del test de *Wilcoxon*, calculamos la r de *Wilcoxon* como medida de tamaño del efecto. Mientras que el valor p determina si una diferencia existe entre dos grupos, la medida del efecto nos dice cuánto impacta o como de grande es esa diferencia:

$$r = \frac{Z}{\sqrt{N}} \tag{25}$$

donde Z es el valor estandarizado del test de *Wilcoxon* [36] y N el número de pares.

Para interpretar el valor de la r de *Wilcoxon*, se definen los rangos en la tabla 1.

r de *Wilcoxon*	Interpretación
$\|r\| < 0{,}1$	Ningún efecto / Efecto muy pequeño
$\|r\| = 0{,}1$	Efecto pequeño
$\|r\| = 0{,}3$	Efecto medio
$\|r\| = 0{,}5$	Efecto grande

Tabla 1: Rangos de interpretación de la r en la prueba de *Wilcoxon* [36].

Si *Shapiro-Wilk* no rechaza la normalidad, realizamos una prueba t de Student bajo las mismas premisas que para el test de *Wilcoxon*. La prueba t de Student devolverá el valor p correspondiente y se cuantificará el tamaño del efecto con la d de *Cohen*:

$$d = \frac{\Delta}{s_{\text{pooled}}} \tag{26}$$

siendo Δ la media de la diferencia definida anteriormente y s_{pooled} la desviación estándar combinada de ambos modelos.

Al igual que para la r de *Wilcoxon*, se definen rangos para interpretar el valor de la d de *Cohen* en la tabla 2.

d de *Cohen*	Interpretación
$d < 0{,}2$	Ningún efecto / Efecto muy pequeño
$d = 0{,}2$	Efecto pequeño
$d = 0{,}5$	Efecto medio
$d = 0{,}8$	Efecto grande

Tabla 2: Rangos de interpretación del tamaño de efecto d de *Cohen* [37].

2.12. El ruido *Perlin*

El ruido *Perlin* [38] es una función matemática que genera una superficie continua asignando a cada vértice de una cuadrícula un vector de gradiente aleatorio. La función toma como entrada pares de coordenadas (x, y), cada una de las cuales se encuentra en una cuadrícula como se muestra en la figura 13.

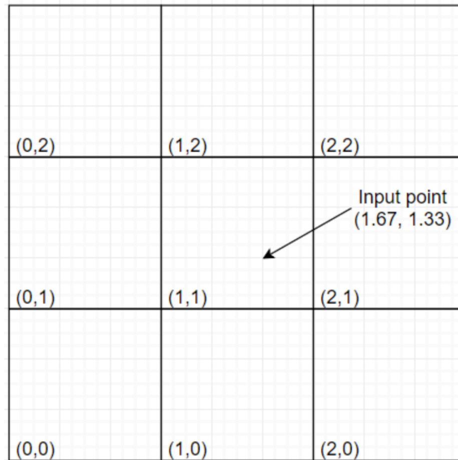


Figura 13: Cuadrícula de valores de entrada [39].

Para cada una de las cuatro esquinas del cuadrado en el que se encuentra el valor de entrada, se asignan dos vectores, uno de ellos apunta de la esquina al valor de entrada y el otro es un vector constante, cuyo gradiente depende de la semilla usada al definir la función. Gráficamente se representa como en la figura 14. Se calcula el producto escalar entre los dos vectores de cada esquina para generar en total cuatro valores.

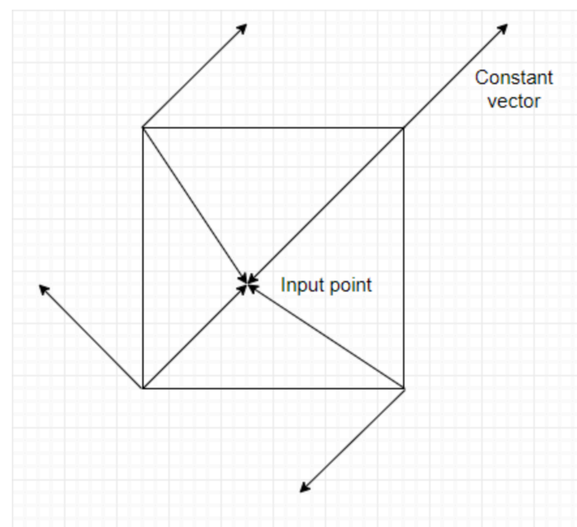


Figura 14: Vectores aleatorios y constantes [39].

A continuación se interpolan los valores de las esquinas superior e inferior izquierda para obtener un valor $v1$. Se realiza el mismo procedimiento para las esquinas superior e inferior derecha para obtener el valor $v2$. Por último, se interpola $v1$ y $v2$, obteniendo el valor final que devolverá la función.

La interpolación en esta función usa una curva llamada *ease curve* o *fade function*, en vez de interpolar linealmente, que produciría transiciones abruptas. Esta curva tiene la forma $6x^5 - 15x^4 + 10x^3$ y se representa gráficamente como en la figura 15.

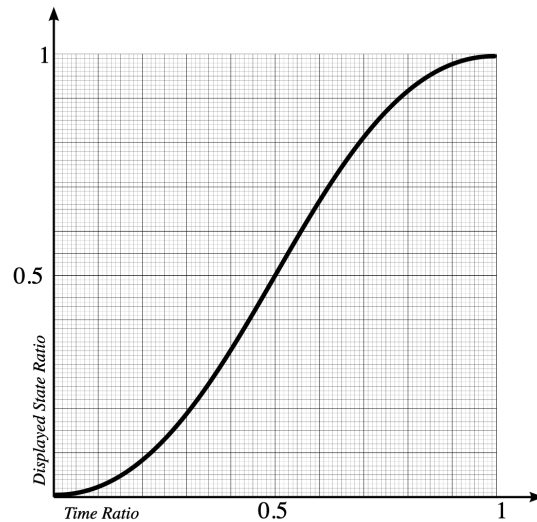


Figura 15: Ease curve o fade function [39].

La derivada de esta curva es cero en los puntos $x = 0$ y $x = 1$, por lo que suaviza la unión entre celdas. Para valores de $x < 0{,}5$, se acerca ligeramente a 0 y para $x > 0{,}5$ se acerca ligeramente a 1, manteniendo $f(0{,}5) = 0{,}5$. El resultado es una transición continua y curvada, que da como resultado la salida que se representa en la figura 16.

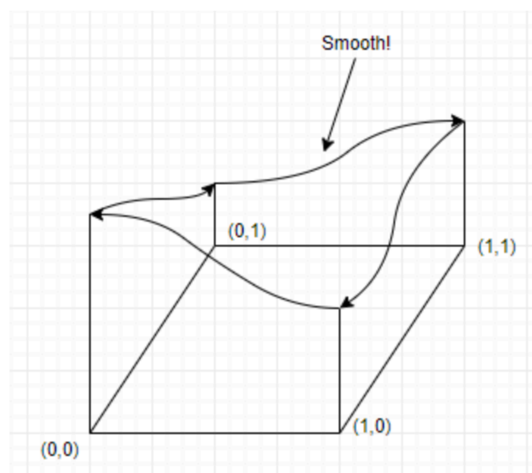


Figura 16: Resultado de la interpolación con ease curve [39].

Tras repetir este proceso para toda la cuadrícula, obtenemos una superficie con transiciones suaves, sin ángulos bruscos, perfecto para simular grietas y erosiones naturales en obras

de arte. En la figura 17 se muestra un ejemplo de la salida de la función de ruido *Perlin*.

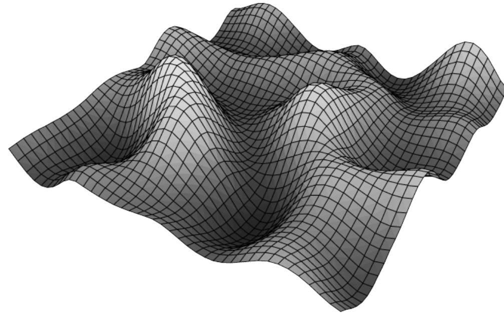


Figura 17: Ejemplo del resultado de la función de ruido *Perlin* [40].

3
Diseño del Sistema

A continuación se describe el proceso seguido en el desarrollo del sistema, comenzando por la creación del *dataset* de entrenamiento, la implementación y evaluación de los distintos modelos propuestos, y finalizando con la obtención del modelo definitivo.

El objetivo principal es desarrollar un modelo de *inpainting* que, además de reconstruir zonas deterioradas, sea capaz de preservar el estilo artístico original de la obra. Para ello, el modelo debe aprender las características inherentes de las diferentes corrientes artísticas, consiguiendo reconstrucciones lo más fieles posibles a cada estilo. En este trabajo se han seleccionado tres estilos: el impresionismo, la iconografía y el arte abstracto. Se entrenará un modelo independiente para cada uno de estos *datasets*, analizando el comportamiento de los modelos ante variaciones estilísticas significativas. Además, se estudiará el impacto del ajuste de los hiperparámetros, adaptándolos en función de las particularidades de cada estilo, con el fin de optimizar la calidad de las reconstrucciones obtenidas.

3.1. Creación del *dataset* de entrenamiento

Por norma general, para el entrenamiento de un modelo de *inpainting* se suelen necesitar decenas de miles de imágenes cuando el dominio no está bien definido. Es decir, si las escenas de las imágenes son drásticamente diferentes, el modelo necesitará más ejemplos para aprender a generalizar correctamente. En este caso, el dominio es específico y controlado, ya que se ha optado por entrenar un modelo independiente para cada estilo artístico y por lo tanto disponiendo también de un *dataset* por estilo.

Sin embargo, no existe actualmente ningún *dataset* de calidad que contenga suficientes ejemplos de obras de arte deterioradas. Por ello, se ha creado un *dataset* propio con daños generados digitalmente mediante un *script* desarrollado en *Python* a partir de un *dataset* con las obras intactas. Este *script* define un *pipeline* de tres etapas, comenzando con la aplicación

de ruido *Perlin* a la imagen.

(a) Polígonos de ángulos aleatorios

(b) Ruido *Perlin*

Figura 18: Comparación entre una máscara binaria generada con polígonos aleatorios o ruido *Perlin*.

Inicialmente se valoró generar máscaras de daños con polígonos de ángulos aleatorios, sin embargo, como se aprecia en la figura 18, el ruido *Perlin* simula daños de manera más natural, por lo tanto posteriormente se decidió cambiar la manera en la que se generan las máscaras a este enfoque.

La segunda etapa del *pipeline* consiste en la generación de grietas, pequeñas fracturas o fisuras sobre la superficie de la obra. El algoritmo genera trayectorias formadas por segmentos conectados, los cuales se dibujan sobre la imagen original con un grosor variable y una coloración próxima a tonos blanquecinos o desaturados, imitando así grietas reales. Estas trayectorias se construyen de manera aleatoria, variando el número de segmentos, su orientación y longitud, obteniendo así grietas distintas en la misma obra.

Este paso del *pipeline* aporta variedad a los ejemplos que el modelo utilizará para aprender. Además, al superponer las grietas sobre las zonas ya alteradas por el ruido *Perlin*, se obtiene una combinación que se acerca aún más al aspecto que puede presentar una obra de arte dañada. Algunos ejemplos del resultado de la aplicación del ruido *Perlin* se muestran en la figura 19.

Por último, aunque no forma parte del *script* mencionado, sino que forma parte del código del modelo, antes del comienzo del entrenamiento se realiza un proceso de *data augmentation* como medida preventiva para mejorar la capacidad de generalización del modelo. En este

(a) Obra de Van Gogh

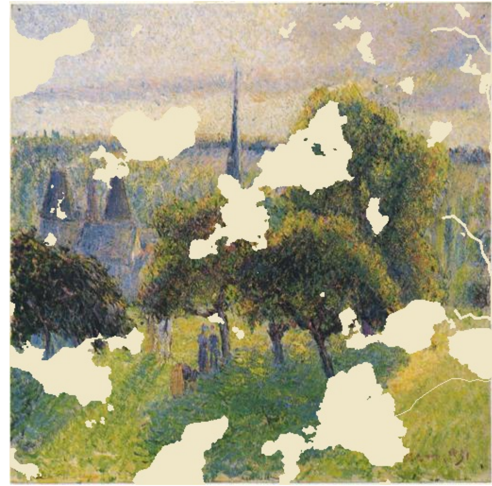

(b) Obra de Pissarro

Figura 19: Ejemplos del *dataset* de artistas impresionistas dañado digitalmente con el *script*.

proceso se realizan pequeñas rotaciones de hasta 5 grados, desplazamientos horizontales y verticales de hasta un 5 %, aumentos de zoom de hasta un 10 % e inversiones horizontales, tal y como se define en la tabla 3. Este proceso se ha seguido para los *datasets* de las tres corrientes

Transformación	**Descripción**
rotation_range=5	Rotación aleatoria en un rango de ±5 grados
width_shift_range=0.05	Desplazamiento horizontal aleatorio de hasta un 5 %
height_shift_range=0.05	Desplazamiento vertical aleatorio de hasta un 5 %
zoom_range=0.1	Zoom aleatorio en un rango de ±10 %
horizontal_flip=True	Inversión horizontal aleatoria (espejado)

Tabla 3: Transformaciones aplicadas durante el *data augmentation*

artísticas mencionadas anteriormente: impresionismo, iconografía y arte abstracto.

3.2. Primera aproximación: Autoencoders

Como primer acercamiento a la tarea de *inpainting*, se optó por utilizar un *Autoencoder* clásico. La primera versión del Autoencoder tenía las características de la tabla 4 y 5, con una entrada de tamaño 512×512 y 3 canales (RGB), $(512, 512, 3)$, normalizada al rango $[0, 1]$ antes de entrar a la red:

Capa	**Filtros**	**Tamaño del *kernel***	**Activación**	**Padding**	**Salida**
Conv2D_1	32	(3,3)	ReLU	same	(512,512,32)
MaxPool2D_1	–	(2,2)	–	same	(256,256,32)
Conv2D_2	64	(3,3)	ReLU	same	(256,256,64)
MaxPool2D_2	–	(2,2)	–	same	(128,128,64)
Conv2D_3	128	(3,3)	ReLU	same	(128,128,128)
MaxPool2D_3	–	(2,2)	–	same	(64,64,128)

Tabla 4: Características del **Encoder**

Capa	**Filtros**	**Tamaño del *kernel***	**Activación**	**Padding**	**Salida**
Conv2D_4	128	(3,3)	ReLU	same	(64,64,128)
UpSampling2D_1	–	(2,2)	–	–	(128,128,128)
Conv2D_5	64	(3,3)	ReLU	same	(128,128,64)
UpSampling2D_2	–	(2,2)	–	–	(256,256,64)
Conv2D_6	32	(3,3)	ReLU	same	(256,256,32)
UpSampling2D_3	–	(2,2)	–	–	(512,512,32)
Conv2D_7	3	(3,3)	Sigmoid	same	(512,512,3)

Tabla 5: Características del **Decoder**

En este modelo se usa la función de activación *ReLU* en las capas internas y sigmoide en la capa de salida.

En la capa de salida se utiliza la función sigmoide ya que comprime los valores al rango [0,1]. Esta característica la hace adecuada para representar las imágenes normalizadas que se usan a la entrada de la red, donde cada canal de color se encuentra dentro de ese mismo rango.

Se utilizó el optimizador *Adam* con una tasa inicial de aprendizaje de $0{,}001$ y la función de pérdida *MSE*.

Durante las primeras pruebas, la curva de pérdida era muy fluctuante, lo que reflejaba una inestabilidad en el entrenamiento y mostraba posibles signos de *overfitting*, como un bajo error en el conjunto de entrenamiento frente a una pobre generalización en las reconstrucciones. A continuación, en la figura 20 se muestran algunos resultados de esta primera iteración:

Figura 20: Resultados de la primera iteración del *Autoencoder*.

Se puede apreciar que las imágenes reconstruidas tienen un ligero tinte no presente en las originales. Este velo azul aparece, entre otras razones, porque el *Autoencoder* se sobreajusta al conjunto de entrenamiento. Al ver siempre las mismas orientaciones y encuadres, la forma más simple y eficaz para minimizar el error es "promediar" los colores dominantes del *dataset*, que en el caso de las obras impresionistas corresponde al canal azul. De esta manera, el modelo genera reconstrucciones con un tono azulado uniforme, independientemente del contenido local.

Para romper este atajo estadístico que el modelo había aprendido, se introdujo un proceso de *data augmentation* que aplicaba las transformaciones mencionadas en la tabla 3. Esta nueva diversidad artificial altera las entradas lo suficiente como para que el modelo ya no pueda seguir apoyándose en un único color dominante. Con ello, la distribución cromática se vuelve más equilibrada y el sesgo hacia el canal azul deja de ser una solución viable.

Por otro lado, al usar *MSE* como función de pérdida, el modelo minimiza las diferencias píxel a píxel entre la imagen original y la reconstruida. El problema de fondo es que el *MSE* no captura la estructura visual ni la percepción humana de calidad, dos imágenes pueden diferir mucho en textura, pero si comparten un canal cromático dominante, presentarán un *MSE* bajo. En este caso, saturar ligeramente el canal azul reduce el *MSE* aunque distorsione el aspecto

real de la obra.

Por ello se decidió sustituir el *MSE* por *SSIM (Structural Similarity Index Measure)*, que evalua la similitud entre imágenes considerando aspectos como la luminancia, el contraste y la estructura local. A diferencia del *MSE*, el *SSIM* penaliza bastante las distorsiones que afectan a la coherencia de la imagen, como la pérdida de curvas, las texturas deformadas o los colores inconsistentes.

Los resultados de siguiente iteración del *Autoencoder* que incluía estos cambios se muestran en la figura 21.

Figura 21: Resultados de la segunda iteración del *Autoencoder*.

La característica más notable de las reconstrucciones es que la reconstrucción tiene un aspecto borroso, como si de una mancha en la obra se tratara, más parecido a un difuminado que a una reconstrucción basada en el contexto. Este comportamiento es una limitación conocida de los *Autoencoders* convencionales, tienden a suavizar las regiones faltantes en lugar de generar detalles coherentes con la imagen original.

Estos últimos resultados podrían indicar una necesidad de pasar a un tipo de modelo más especializado para la tarea de *inpainting*, pero primero se intenta introducir dos nuevas características al modelo: una reducción del cuello de botella y la arquitectura *U-Net*. La reducción del cuello de botella fue una modificación inspirada en el artículo *A Generative Image Inpain-*

ting Model Based on Edge and Feature Self-Arrangement Constraints [41]. En este artículo se recomienda reducir el cuello de botella a una representación espacial mínima, idealmente de tamaño 1×1. Sin embargo, el tamaño de las imágenes de entrada en el modelo del artículo es de 256×256 y en nuestro modelo la entrada es de tamaño 512×512, con la finalidad de preservar todo el detalle posible. Por ello, se procedió con una reducción a 4×4, añadiendo así dos capas más al modelo.
Por otro lado, la motivación tras la introducción de la arquitectura *U-Net* se debe a que esta estrategia ayuda a preservar bordes, contornos y otras estructuras locales que de otro modo podrían perderse al pasar por el cuello de botella.

La nueva estructura es:

Capa	**Filtros**	**Tamaño del *kernel***	**Activación**	**Padding**	**Salida**
Conv2D (c1)	32	(3,3)	ReLU	same	(512,512,32)
MaxPool2D (p1)	–	(2,2)	–	same	(256,256,32)
Conv2D (c2)	64	(3,3)	ReLU	same	(256,256,64)
MaxPool2D (p2)	–	(2,2)	–	same	(128,128,64)
Conv2D (c3)	128	(3,3)	ReLU	same	(128,128,128)
MaxPool2D (p3)	–	(2,2)	–	same	(64,64,128)
Conv2D (c4)	256	(3,3)	ReLU	same	(64,64,256)
MaxPool2D (p4)	–	(2,2)	–	same	(32,32,256)
Conv2D (c5)	512	(3,3)	ReLU	same	(32,32,512)
MaxPool2D (p5)	–	(2,2)	–	same	(16,16,512)
Conv2D (c6)	512	(3,3)	ReLU	same	(16,16,512)
MaxPool2D (p6)	–	(2,2)	–	same	(8,8,512)
Conv2D (c7)	512	(3,3)	ReLU	same	(8,8,512)
MaxPool2D (p7)	–	(2,2)	–	same	(4,4,512)

Tabla 6: Características del **Encoder**

Capa	Filtros	Tamaño del *kernel*	Activación	Padding	Salida
UpSampling2D (u1)	–	(2,2)	–	–	(8,8,512)
Conv2D (d1)	512	(3,3)	ReLU	same	(8,8,512)
Concatenate (m1)	–	–	–	–	(8,8,1024)
UpSampling2D (u2)	–	(2,2)	–	–	(16,16,1024)
Conv2D (d2)	512	(3,3)	ReLU	same	(16,16,512)
Concatenate (m2)	–	–	–	–	(16,16,1024)
UpSampling2D (u3)	–	(2,2)	–	–	(32,32,1024)
Conv2D (d3)	512	(3,3)	ReLU	same	(32,32,512)
Concatenate (m3)	–	–	–	–	(32,32,1024)
UpSampling2D (u4)	–	(2,2)	–	–	(64,64,1024)
Conv2D (d4)	256	(3,3)	ReLU	same	(64,64,256)
Concatenate (m4)	–	–	–	–	(64,64,512)
UpSampling2D (u5)	–	(2,2)	–	–	(128,128,512)
Conv2D (d5)	128	(3,3)	ReLU	same	(128,128,128)
Concatenate (m5)	–	–	–	–	(128,128,256)
UpSampling2D (u6)	–	(2,2)	–	–	(256,256,256)
Conv2D (d6)	64	(3,3)	ReLU	same	(256,256,64)
Concatenate (m6)	–	–	–	–	(256,256,128)
UpSampling2D (u7)	–	(2,2)	–	–	(512,512,128)
Conv2D (d7)	32	(3,3)	ReLU	same	(512,512,32)
Concatenate (m7)	–	–	–	–	(512,512,64)
Conv2D (decoded)	3	(3,3)	Sigmoid	same	(512,512,3)

Tabla 7: Características del **Decoder** con conexiones de salto

El resultado tras estos cambios se muestran en la figura 22.

Figura 22: Resultados de la segunda iteración del *Autoencoder*.

A pesar de las mejoras introducidas, los resultados visuales siguen siendo subóptimos. Las zonas reconstruidas continúan presentando un parche difuso y poco detallado, lo que indica que el modelo no es capaz de generar contenido visual realista en regiones faltantes. Por ello, en la siguiente fase se exploró el uso de *GANs*, diseñados específicamente para producir imágenes más coherentes y nítidas.

3.3. Segunda aproximación: *GANs*

La siguiente aproximación se basa en redes generativas adversarias (*GANs*), donde el *Autoencoder* desarrollado anteriormente actuará como el generador. De esta manera, se espera que el parche que las reconstrucciones mostraban en el *Autoencoder* mejoren, ya que el discriminador forzará al generador a crear imágenes más realistas.

La estructura del modelo *GAN* implementado es la siguiente. Por simplicidad, se muestran los detalles de las capas del discriminador, puesto que como se ha mencionado anteriormente, la arquitectura del generador es la misma que la arquitectura del *Autoencoder*, mostrada en las tablas 6 y 7:

Capa	**Filtros**	**Tamaño del *kernel***	**Activación**	**Padding**	**Salida**
Conv2D (1)	64	(4,4)	LeakyReLU ($\alpha = 0.2$)	same	(256,256,64)
Conv2D (2)	128	(4,4)	LeakyReLU ($\alpha = 0.2$)	same	(128,128,128)
Conv2D (3)	256	(4,4)	LeakyReLU ($\alpha = 0.2$)	same	(64,64,256)
Conv2D (Salida)	1	(4,4)	Lineal	same	(64,64,1)

Tabla 8: Características del **Discriminador (PatchGAN)**

La estructura del discriminador está basada en la implementación del *GAN pix2pix* [42].

Se utilizan convoluciones con un *kernel* de tamaño 4x4, siguiendo el ejemplo del modelo *pix2pix*. En este tamaño se encuentra el equilibrio entre la capacidad de capturar patrones y el coste computacional. Un *kernel* demasiado pequeño, como 1x1, sería muy limitado y apenas percibiría la textura y la estructura de los parches de imagen, mientras que un *kernel* más grande, como 5x5 o mayor, multiplicaría el número de parámetros y, por tanto, la carga de cálculo y memoria, sin necesariamente mejorar la captura de patrones. Se combina con un *stride* de 2 en las tres primeras capas.

Por otro lado, el uso de la función de activación *Leaky ReLU* en lugar de la versión estándar se ha elegido para mejorar la estabilidad del entrenamiento y evitar el problema del *dying ReLU*. En una red como el discriminador de un *GAN* existe el riesgo de que muchas neuronas terminen produciendo cero de forma constante, lo que impide que vuelvan a aprender en iteraciones posteriores. Con la función de activación *Leaky ReLU*, en la que introducimos una pendiente de $0{,}2$, evitamos que las neuronas no se apaguen por completo.

En un *GAN* el equilibrio entre el generador y el discriminador es delicado, puede verse fácilmente afectado por saturaciones, por lo que la función *Leaky ReLU* hace más robusto el entrenamiento en estas redes. Este *GAN* utiliza como función de pérdida tanto *binary-cross entropy* (*BCE*) como *L1 loss*. Esta es la formulación original del *GAN pix2pix*, el discriminador usa *BCE* para aprender a distinguir imágenes verdaderas de falsas y el generador utiliza una combinación de *BCE* y *L1 loss*.

Algunos de los resultados obtenidos con 50 épocas de entrenamiento se muestran en la figura 23.

A continuación se realizó un experimento de entrenamiento a 80 épocas para asegurar que

Figura 23: Resultados de la primera iteración del *GAN*.

el modelo no estaba siendo infraajustado con las 50 épocas de las pruebas anteriores. Algunos resultados se muestran en la figura 24. Observamos en la figura 25 que, tras aproximadamente 40-50 épocas, la función de pérdida se estabiliza y no mejora de forma significativa, lo que confirma que el desempeño del modelo no estaba limitado por el número de épocas en los experimentos anteriores.

Para mejorar la reconstrucción, se sustituye el *L1 loss* por la función *SSIM*, que como se ha mencionado antes, captura mejor las texturas y estructuras desde un punto de vista humano.

Para integrar la función *SSIM* correctamente en el modelo, se deberá ajustar el rango de salida de *SSIM*. Inicialmente se minimizaba la pérdida de la ecuación 27.

$$\mathcal{L}\text{rec} = 1 - \text{SSIM}(x, \hat{x}) \tag{27}$$

lo que podía producir valores de $\mathcal{L}$rec tan altos como 2 si SSIM $= -1$. Al combinar esto en la función de coste global, que toma la forma de la ecuación 28.

$$\mathcal{L}global = \mathcal{L}\text{adv} + \lambda\mathcal{L}\text{rec} \tag{28}$$

donde $\mathcal{L}\text{adv} = BCE \in [0, 1]$ y λ es el peso que se le atribuye a la reconstrucción, el término de reconstrucción corría el riesgo de dominar por completo el entrenamiento. Para

Figura 24: Resultados de la segunda iteración del *GAN*.

asegurar que $\mathcal{L}$rec no sobrepasara el limite superior de $[0, 1]$, se redefine la pérdida como se ve en la ecuación 29.

$$\mathcal{L}_{\text{rec}} = \frac{1 - \text{SSIM}(x, \hat{x})}{2}. \tag{29}$$

De este modo, la componente de reconstrucción nunca supera los límites de Ladv y los gradientes mantienen una escala estable.

Finalmente, se revisó el peso de la reconstrucción. Se eligió originalmente $\lambda = 50$, lo cual otorgaba demasiada importancia a la reconstrucción frente al engaño al discriminador, rompiendo el equilibrio del *GAN*. Se redujó entonces λ a 10, de modo que $\mathcal{L}$adv y $\mathcal{L}$rec contribuyeran por igual al descenso de gradiente.

Tras estos cambios, algunos de los resultados se muestran en la figura 26 y la curva de la pérdida se muestra en la figura 27.

Visualmente los resultados son buenos, pero además, se puede apreciar como la pérdida del discriminador converge sobre $1{,}3 - 1{,}35$. Como se comentó anteriormente, se considera que un modelo *GAN* está entrenado cuando el discriminador no es capaz de distinguir entre una imagen real y una falsa, es decir, la salida del discriminador es igual a $0{,}5$.
Usando la fórmula *Min-Max* podemos comprobar que para $D(x) = 0{,}5$ y $D(G(z)) = 0{,}5$:

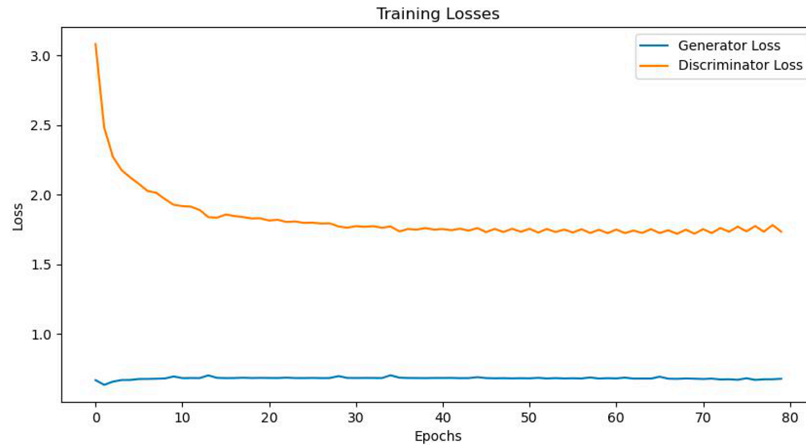


Figura 25: Evolución del valor de la función de pérdida durante el entrenamiento (segunda iteración del *GAN*).

$log(0{,}5) + log(1 - 0{,}5) = -1{,}386$

Es decir, cuando la pérdida del discriminador es $1{,}386$, su precisión es exactamente $0{,}5$. El modelo diseñado se estabiliza sobre este rango, por lo que podemos concluir con que la precisión se encuentra sobre $0{,}5$ también. En conclusión, no solo la reconstrucción es visualmente buena, sino que la curva de la pérdida respalda esta afirmación.

3.4. El modelo ICT

Además de diseñar dos modelos propios, en este trabajo se ha explorado el uso del *Transformer ICT* [1], el cual aborda varias limitaciones propias de las redes convolucionales. Estas limitaciones se hicieron evidentes en nuestros modelos, a continuación se señalan algunos de los aspectos más relevantes que destaca el artículo original del *ICT*:

1. Una convolución se enfoca en patrones locales al rellenar regiones cercanas, por tanto, es complicado reconstruir imágenes que requieren un contexto global completo.

2. Los filtros de una red convolucional son traslacionalmente invariantes, por eso los patrones duplicados o los artefactos borrosos aparecen con frecuencia en las regiones donde se aplican las máscaras de daño.

El artículo propone una solución que combina las fortalezas de los *Transformers* y de las redes convolucionales:

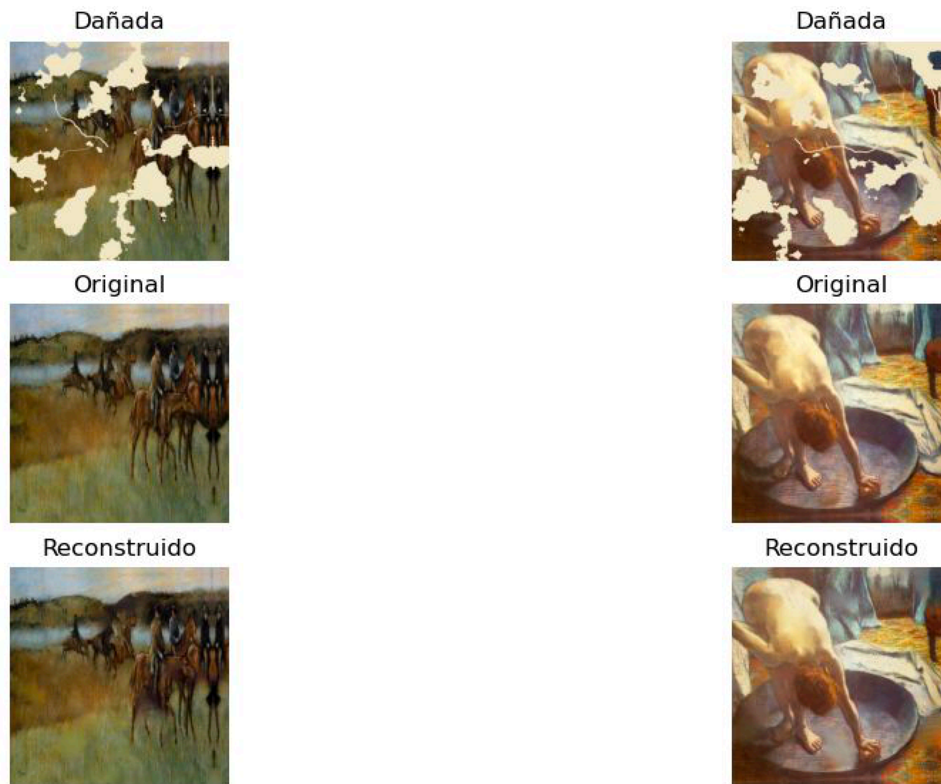


Figura 26: Resultados de la tercera iteración del *GAN*.

1. Un *Transformer* que genere una reconstrucción a baja resolución, aprovechando su capacidad para entender el contexto global y, al mismo tiempo, debido a su naturaleza estocástica, introducir no determinismo para producir varias soluciones plausibles.

2. Una red convolucional basada en la arquitectura de los *Autoencoders* que, a partir de la salida de baja resolución del *Transformer*, refine los detalles locales y elimine parches borrosos.

Los resultados del *ICT* fueron buenos desde el primer momento, tal y como se muestra en las figuras 28 y 29, por lo tanto no se realizaron cambios de diseño reseñables.

El modelo *ICT*, debido a su naturaleza estocástica, puede generar varias reconstrucciones diferentes de una misma imagen, sin embargo, en este trabajo se ha optado por producir una única reconstrucción.

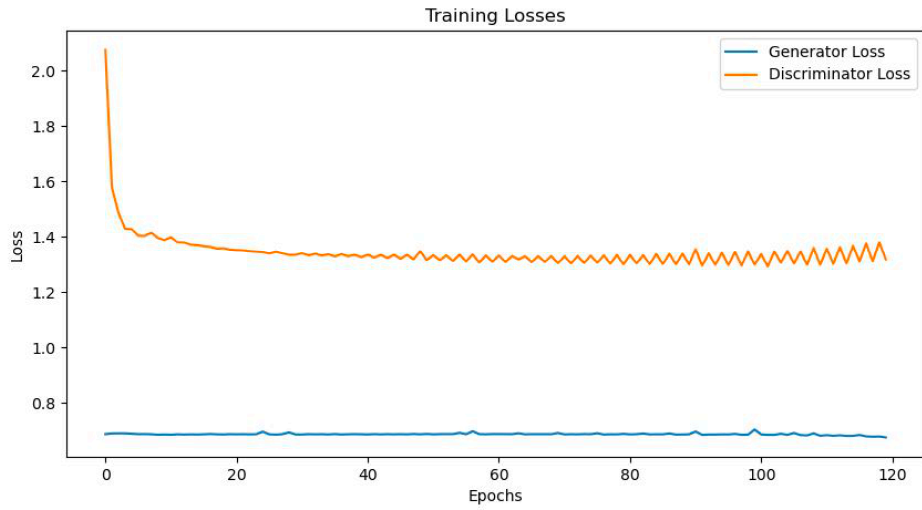


Figura 27: Recta de pérdida de la tercera iteración del *GAN*.

(a) Original (b) Dañada (c) Reconstruida

Figura 28: Ejemplo de reconstrucción con *ICT*.

(a) Original (b) Dañada (c) Reconstruida

Figura 29: Ejemplo de reconstrucción con *ICT*.

4

Análisis de los resultados

Una vez terminado el desarrollo del modelo *GAN* y puesto en funcionamiento el *ICT*, se procedió primero al estudio cualitativo, que consistió en la evaluación visual directa de las reconstrucciones.

A continuación, se llevó a cabo el estudio cuantitativo de los resultados, con el objetivo de aportar métricas objetivas que complementen la percepción subjetiva que una persona pueda tener de las imágenes reconstruidas.

4.1. Estudio cualitativo

A continuación se mostrarán ejemplos de reconstrucciones generadas por el modelo *GAN* y por el *ICT* aplicados a la misma imagen de entrada, de modo que pueda compararse su desempeño y determinar cuál de ellos produce mejores resultados.

Para el conjunto de datos de arte impresionista, las diferencias entre los modelos se ilustran en las figuras 30 y 31. Para el conjunto de datos de arte abstracto, la comparación aparece en las figuras 32 y 33. Finalmente, para el conjunto de datos de iconografía, los resultados se muestran en las figuras 34 y 35.

Para el conjunto de arte impresionista, el *ICT* devuelve mejores resultados visualmente. Una de las razones para esto se debe a su mecanismo de atención contextual global, que es capaz de conectar patrones alejados en la imagen. Esto mejora la coherencia de los bordes y logra transiciones de color más naturales.

Esto es especialmente importante para esta corriente, donde las escenas representadas buscan plasmar la luz y el instante, sin importar demasiado la identidad de aquello que se proyecta. Los objetos no se definen, sino que se pinta su impresión visual y eso implica que las partes

(a) Original

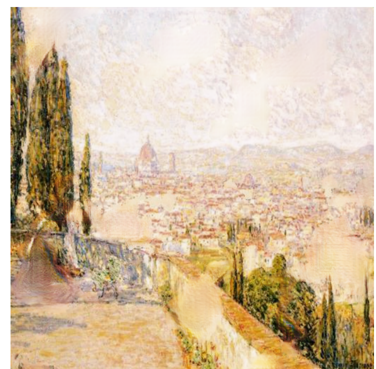
(b) Dañada

(c) Reconstruida

Figura 30: Ejemplo de reconstrucción del *ICT* para arte impresionista.

(a) Original

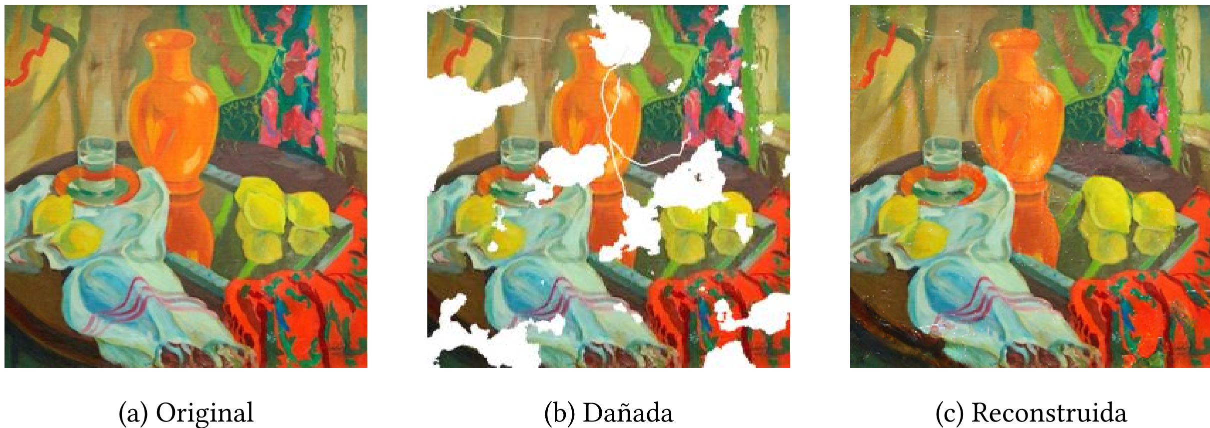
(b) Dañada



(c) Reconstruida

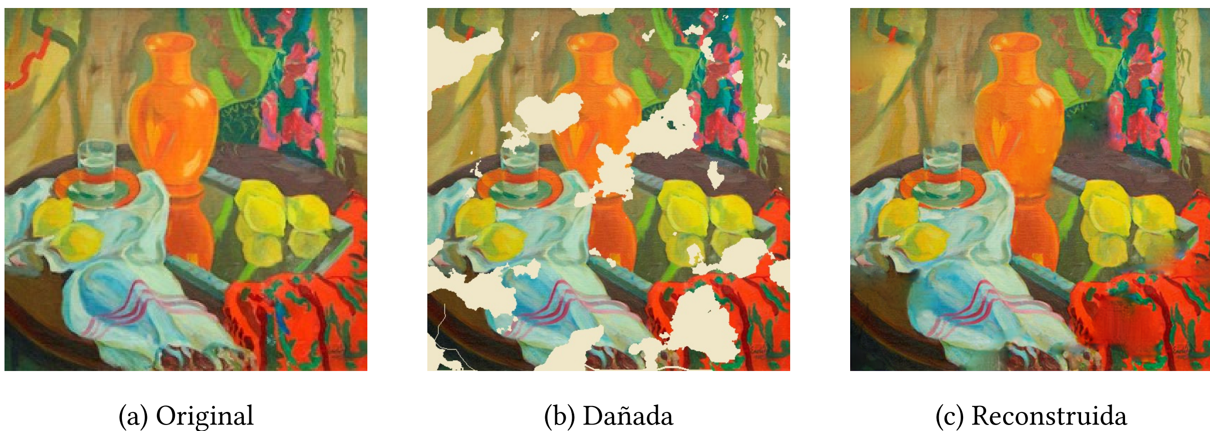
Figura 31: Ejemplo de reconstrucción del *GAN* para arte impresionista.

inconexas dan lugar a un todo unitario [43]. Para un *GAN*, cuyo rendimiento es mejor cuando las líneas son nítidas y las texturas están bien definidas, abordar el inpainting en escenas impresionistas se convierte en un desafío especialmente complejo.



(a) Original (b) Dañada (c) Reconstruida

Figura 32: Ejemplo de reconstrucción del *ICT* para arte abstracto.



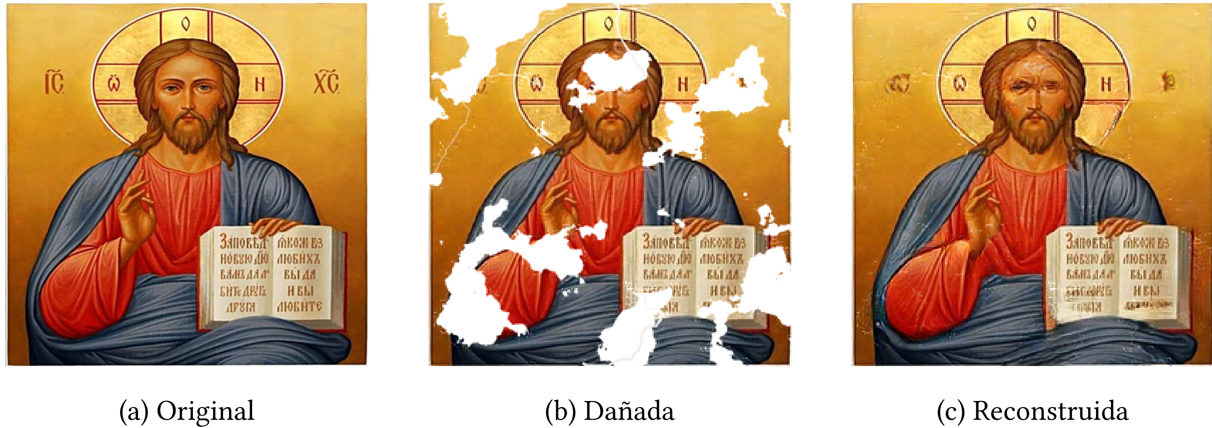
(a) Original (b) Dañada (c) Reconstruida

Figura 33: Ejemplo de reconstrucción del *GAN* para arte abstracto.

Por otro lado, para el conjunto de arte abstracto no se aprecia mejoría entre ambos modelos. Esto se debe a que esta corriente se centra más en la utilización de formas geométricas simples, con un énfasis en las texturas de la obra y bloques de color relativamente homogéneos [44]. Las dependencias espaciales a larga distancia no son tan comunes como en el arte impresionista y el detalle en la obra es menor, por lo que un *GAN* convencional puede reproducir estas estructuras locales con la misma eficacia que el *ICT*.

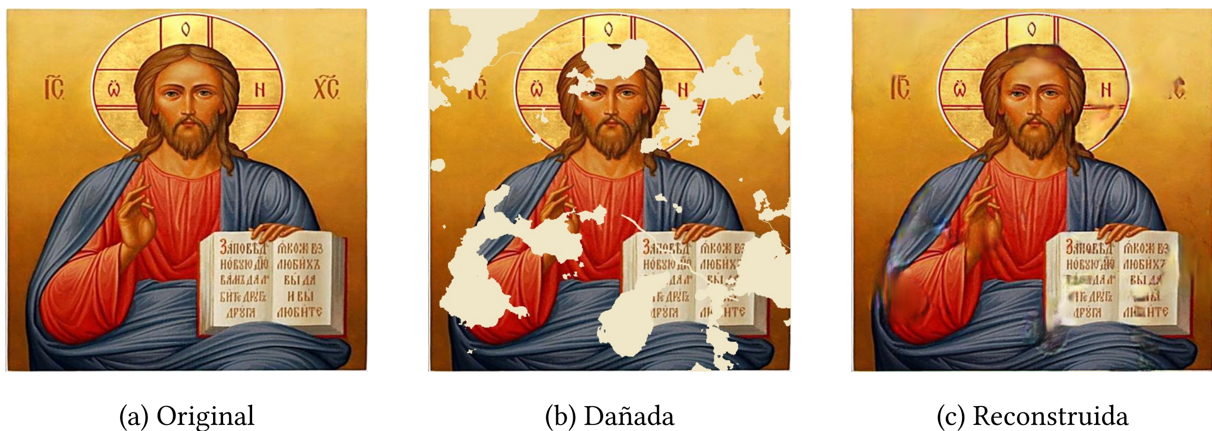
Además, la heterogeneidad estilística y la ausencia de patrones consistentes en el arte abstracto dificultan que el mecanismo de atención del *ICT* encuentre relaciones en las imágenes,

por lo que su complejidad arquitectónica comparado con el *GAN* no se traduce en una mejora perceptible.



(a) Original (b) Dañada (c) Reconstruida

Figura 34: Ejemplo de reconstrucción del *ICT* para iconografía.



(a) Original (b) Dañada (c) Reconstruida

Figura 35: Ejemplo de reconstrucción del *GAN* para iconografía.

Por último, en el conjunto de iconografía se aprecian patrones reiterativos y formas geométricas sencillas. Al igual que en el arte abstracto, donde el protagonista central son las figuras geométricas básicas, la iconografía sitúa un motivo principal en sus obras, una figura religiosa generalmente rodeada de elementos geométricos bien definidos. Estos rasgos se captan fielmente gracias que las convoluciones capturan de forma muy precisa detalles como contornos nítidos, mientras que la menor necesidad de contexto global reduce la ventaja del mecanismo de atención del *ICT*.

Por lo tanto, la simplicidad del *GAN* resulta más adecuada para reproducir los bordes nítidos y las texturas uniformes que caracterizan este estilo.

4.2. Estudio cuantitativo

Se comenzó por recopilar las métricas definidas en el capítulo de conceptos preliminares:

MSE	SSIM	LPIPS
DISTS	FSIM	ORB

Estas métricas se organizaron en dos archivos *csv*, uno para los resultados del *GAN* y otro para el *ICT*, para así facilitar su conversión a *dataframe*, que es una estructura de datos para el análisis de datos, en el script de *Python* que calculará el error absoluto, el test de *Wilcoxon* y la d de *Cohen*.

Imagen	MSE	SSIM	ORB	FSIM	LPIPS	DISTS
243859.jpg	0.0048533	0.7852006	316	0.8949206	0.1512046	0.1093499
243899.jpg	0.0072402	0.8186990	340	0.9048578	0.1441420	0.0989689
⋮	⋮	⋮	⋮	⋮	⋮	⋮
244042.jpg	0.0016752	0.8339232	371	0.9234622	0.1588286	0.0812939

Tabla 9: Métricas del modelo *ICT* para un subconjunto de imágenes.

Imagen	MSE	SSIM	ORB	FSIM	LPIPS	DISTS
244390.jpg	0.0060775	0.8292871	346	0.9110358	0.2044860	0.1150157
244384.jpg	0.0031514	0.8650702	317	0.9326500	0.2163009	0.1084991
⋮	⋮	⋮	⋮	⋮	⋮	⋮
244391.jpg	0.0028743	0.8746661	345	0.9391953	0.1891468	0.0844485

Tabla 10: Métricas del modelo *GAN* para un subconjunto de imágenes.

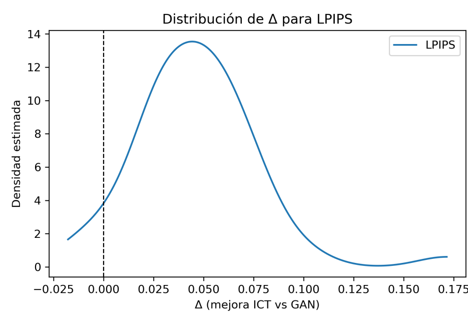
Cabe destacar que, para las métricas *MSE*, LPIPS y *DISTS*, un valor menor indica mejor desempeño, mientras que para *SSIM*, *FSIM* y *ORB*, valores más altos son preferibles.

Los resultados para el test de normalidad de *Shapiro-Wilk* fueron los siguientes:

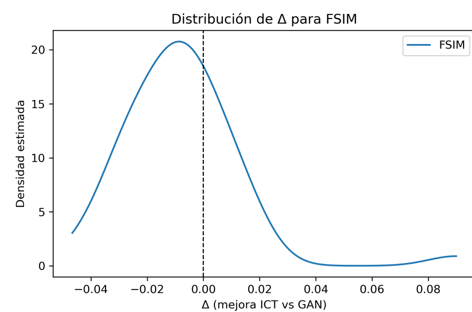
Métrica	**p-valor**
LPIPS	0.003
DISTS	0.000
FSIM	0.000
SSIM	0.840
MSE	0.357

Tabla 11: Resultados del test de normalidad *Shapiro-Wilk*

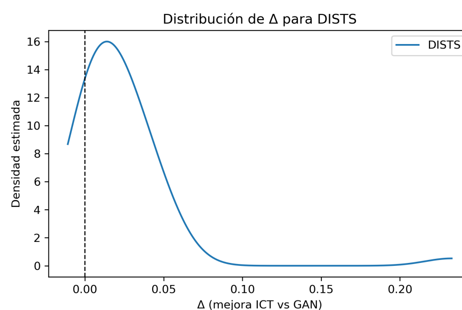
Para este test los decimales se recortan ya que solo son necesarios los primeros 2 decimales. Como vemos, para las métricas *DISTS*, *FSIM* y LPIPS, $p < 0{,}05$, por lo tanto podemos decir que no siguen una distribución normal.



(a) Curva LPIPS



(b) Curva FSIM



(c) Curva DISTS

Figura 36: Comparación de curvas LPIPS, FSIM y DISTS.

Sus curvas muestran un repunte en la cola derecha, como se puede apreciar en la figura 36, que es suficiente para romper la distribución normal.

Por lo tanto, para estas métricas se utilizará el test de *Wilcoxon*, que no tiene como requisito

una distribución normal de los datos para su uso, mientras que para *MSE*, *SSIM* y *ORB* se usará la d de *Cohen*.

El resultado de las pruebas se muestran en la tabla 12.

Métrica	$\mathbf{\Delta}$	**Cohen's** d	p_{tStudent}	r_{Wilcoxon}	p_{Wilcoxon}
LPIPS	0.0471	1.4482	$8{,}10 \times 10^{-13}$	0.8624	$2{,}84 \times 10^{-13}$
DISTS	0.0241	0.6471	$4{,}10 \times 10^{-05}$	0.7227	$4{,}81 \times 10^{-08}$
FSIM	-0.0066	-0.3903	0.99396	0.4821	0.99955
SSIM	-0.0229	-1.4536	0.99999	0.8472	0.99999
MSE	0.0016	0.9379	$6{,}29 \times 10^{-08}$	0.8119	$1{,}22 \times 10^{-10}$
ORB	2.3333	0.0448	0.38259	0.0303	0.41949

Tabla 12: Comparación *ICT* vs. *GAN*.

Interpretación de los resultados Para el análisis de los datos se definió como hipótesis nula H_0 que ambos modelos son equivalentes en rendimiento y como hipótesis alternativa H_1, que el *ICT* superaba al *GAN*.

Para *LPIPS*, el modelo *ICT* mostró una reducción promedio de 0.047 puntos respecto al *GAN*. De forma similar, *ICT* también mejoró *DISTS* en 0.024 puntos y *MSE* en aproximadamente 0.002 puntos. Aunque *ICT* incrementó ligeramente *ORB* en 2.33 puntos, esta diferencia no resultó relevante dado el amplio rango de valores de esta métrica $[300, 400]$.

Por otro lado, *FSIM* (-0.0066) y *SSIM* (-0.0229) favorecen al modelo *GAN*.

Sin embargo, únicamente con Δ no es posible llegar a ninguna conclusión, para determinar si estas diferencias son estadísticamente significativas se debe demostrar que estas diferencias no se han dado por azar.

En métricas con distribución normal, se observó mediante la d de *Cohen* un efecto considerable en *MSE* (d = 0.94), confirmando claramente la ventaja del *ICT*, y un efecto muy grande en *SSIM* (d = -1.45), favoreciendo al *GAN*.

En *ORB*, el efecto resultó trivial (d = 0.045). La prueba *t Student* apoyó estos resultados, confirmando que para *MSE* se puede rechazar la hipótesis nula y para *SSIM* no. Descartamos *ORB* ya que no existe evidencia de diferencia.

En las métricas sin distribución normal, los resultados de la prueba de *Wilcoxon* evidenciaron claramente la superioridad del *ICT* en *LPIPS* y *DISTS* (valores altos de r y muy bajos valores p), mientras que *FSIM* mostró una leve ventaja a favor del *GAN* (r = 0.482). Esto de nuevo, se respalda con los valores p, donde para *LPIPS* y *DISTS* se rechaza la hipótesis nula y para *FSIM* (p-$W = 0.\hat{9}$), no se puede rechazar.

Una observación interesante es que para las métricas en las que cuanto más pequeñas son, mejor, el *ICT* tiene ventaja, mientras que para las métricas en las que un valor más grande es mejor, gana el *GAN*. Esto no resulta una simple curiosidad, sino que es esperable. El *ICT* se enfoca en minimizar el error de apariencia y el *GAN* en maximizar la fidelidad estructural, cada modelo brilla en su especialidad.

En conclusión, el modelo *ICT* ofrece mejoras claras tanto visuales como numéricas en las métricas *LPIPS*, *DISTS* y *MSE*, mostrando efectos importantes y resultados estadísticamente significantes. Por otro lado, el modelo *GAN* destaca en la conservación de la estructura local y los contornos, reflejado en las métricas *SSIM* y *FSIM*. Sin embargo, estas ventajas del *GAN* parecen deberse principalmente al mejor contraste en bordes y no necesariamente a una mayor calidad en las texturas semánticas. Finalmente, la métrica *ORB* no mostró diferencias relevantes entre ambos modelos.

5
Uso del Sistema

Para facilitar el uso de los modelos *GAN* e *ICT*, se han desarrollado dos interfaces, usando el framework *Flask*.

Interfaz del *GAN* En esta interfaz se presenta una lista de obras de las tres corrientes artísticas contempladas en este trabajo y un desplegable con el que elegir el modelo concreto con el que se realizará la inferencia.

Al pulsar sobre el botón "Subir", se enviará una solicitud al *script*, que cargará el archivo *.keras* correspondiente al modelo seleccionado. En este archivo se guarda la descripción de las capas del modelo, sus pesos tras el entrenamiento y otros parámetros.

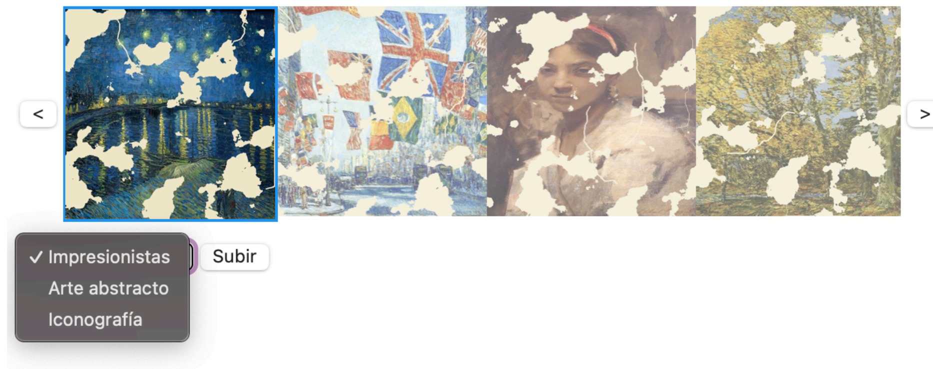
Los resultados se mostrarán junto a las métricas *MSE*, *SSIM*, *FSIM*, *DISTS*, *ORB* y *LPIPS*.

Interfaz del *ICT* Al igual que en la interfaz del *GAN*, se mostrará una lista de obras, junto a una lista adicional en la que se podrá seleccionar el daño a aplicar a la obra. Esta diferencia de diseño se debe al funcionamiento del *script* de inferencia del modelo *ICT*, donde se toma como parámetro la obra original sin dañar y la máscara de daño, aplicando la máscara a posteriori.

Al igual que en el *GAN*, el botón "Subir" enviará una solicitud al *script*, que ejecutará el comando que iniciará la inferencia en el modelo *ICT*.

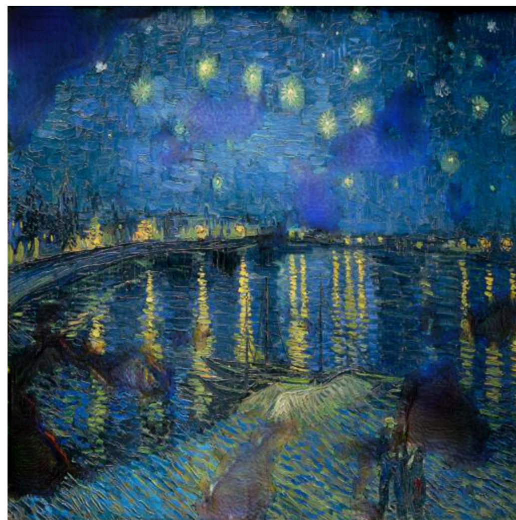
La salida es análoga a la del *GAN*.

Selecciona una imagen de ejemplo:



(a) Interfaz *GAN*.

Reconstrucción completada para impresionistas



Diferencia MSE: 0.005

Similitud SSIM: 0.7789

Numero de similitudes detectadas por ORB: 255

Similitud FSIM: 0.8967

Similitud LPIPS: 0.2346

Similitud DISTS: 0.1361

(b) Resultados en la interfaz del *GAN*.

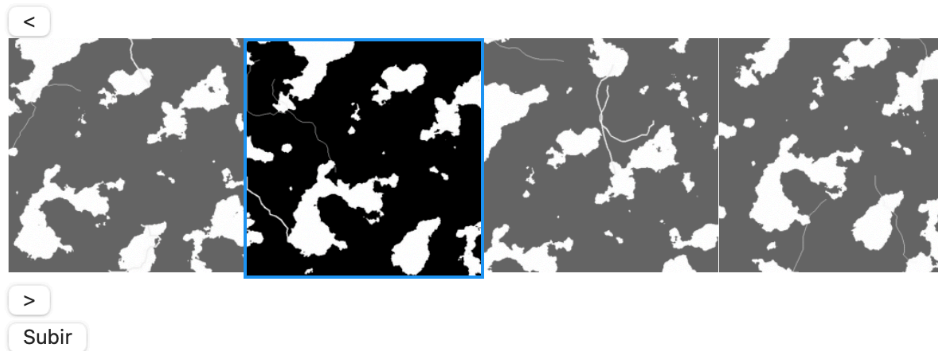
Selecciona una imagen de ejemplo:

<

>

Selecciona una máscara de ejemplo:

<

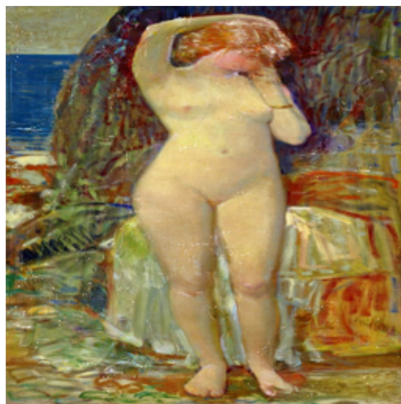


>

Subir

(a) Interfaz *ICT*.

Reconstrucción completada



Diferencia MSE: 0.0026

Similitud SSIM: 0.8224

Similitud FSIM: 0.9078

Similitud LPIPS: 0.1653

Similitud DISTS: 0.1021

Numero de similitudes detectadas por ORB: 256

(b) Resultados en la interfaz del *ICT*.

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

Para terminar este trabajo, se establece que el *ICT* y el *GAN* ofrecen fortalezas complementarias, reflejando tanto las peculiaridades de los *datasets*, como la naturaleza de las métricas con las que los evaluamos. Antes de compararlos, cabe mencionar de nuevo que en el presente trabajo se han entrenado dos modelos de *inpainting*, uno basado en *ICT* y otro diseñado desde cero basado en la arquitectura *GAN*. A continuación, las observaciones más importantes son:

1. *ICT* vence en "menos es mejor" (LPIPS, DISTS, MSE), su entrenamiento con pérdidas orientadas a minimizar directamente ese tipo de errores le permite reconstruir texturas con muy bajo LPIPS y DISTS, y con MSE reducido. Visualmente, se aprecia un acabado más suave y coherente con la referencia.

2. *GAN* vence en "más es mejor" (SSIM, FSIM), los discriminadores adversarios refuerzan contornos nítidos y patrones de alto contraste que elevan estas métricas de similitud estructural. Incluso si a simple vista parecen "borrones", esas salidas reproducen bordes con exactitud pixel a pixel que SSIM y FSIM premian.

Esto nos enseña que no existe un mejor absoluto, cada modelo optimiza un aspecto distinto de la calidad de imagen. El *GAN*, basado en convoluciones locales, iguala al *ICT* en dominios donde predominan bloques de color homogéneos y formas geométricas sencillas, como el arte abstracto o la iconografía. Sin embargo, de cara a minimizar el error perceptual, el *ICT* tiene ventaja gracias a su atención global, que conecta zonas distantes en la obra y reduce artefactos al reconstruir coherentemente texturas definidas vagamente y estructuras extendidas, como en el arte impresionista.

En definitiva, el *ICT* y nuestro *GAN* son dos herramientas que atacan el mismo problema desde dos ángulos distintos. Sin embargo, en el *ICT* la red convolucional implementada es muy parecida a un autoencoder clásico y durante el desarrollo de este trabajo se ha determinado que reformular un autoencoder como un *GAN* potencia la generación de texturas más realistas y detalladas. Por ello, se considera que la clave está en combinar las virtudes del *ICT* con el *GAN* convolucional para lograr restauraciones digitales que integren calidad visual y fidelidad estructural de forma equilibrada.

Durante el proceso de desarrollo he podido profundizar en un ámbito del *software* que me resulta tanto interesante como intimidante. Partí de unos conocimientos muy básicos de inteligencia artificial y mi experiencia con *Python* se limitaba a unos pocos proyectos propios. He tenido que aprender un lenguaje de programación nuevo, usar librerías nuevas y, sobre todo, acostumbrarme a leer y comprender artículos científicos para que mi entendimiento sobre los modelos usados no quedara en la superficie. Todo esto me ha dado una base sólida, tanto práctica como teórica, en este campo tan interesante, al cual me encantaría dedicarme en mi futura vida profesional.

6.2. Líneas Futuras

Como línea futura se propone sustituir el bloque del *ICT* correspondiente a la red convolucional por el *GAN* desarrollado en este trabajo, de este modo, el *ICT* seguiría aprovechando su atención global para recuperar estructuras coherentes a larga distancia, mientras que la presión adversarial aseguraría una generación local de texturas y bordes aún más nítido y fiel al estilo original.

También se propone estudiar arquitecturas alternativas para el modelo *GAN*, con el fin de comprobar si superan en rendimiento a la configuración actual formada por la arquitectura *U-Net* y el discriminador *pix2pix*.

Por último, una posible extensión al trabajo realizado es la aplicación del modelo *GAN* desarrollado a la restauración de centros históricos dañados.

Referencias

[1] Ziyu Wan, Jingbo Zhang, Dongdong Chen, and Jing Liao. High-fidelity pluralistic image completion with transformers. *arXiv preprint arXiv:2103.14031*, 2021. Accedido: 15 de marzo de 2025.

[2] Rijksmuseum. Operation night watch. https://www.rijksmuseum.nl/en/stories/operation-night-watch?filter=Research%20%20treatment, 2019. Accedido: 12 de marzo de 2025.

[3] IBM Technology. What are autoencoders? https://www.youtube.com/watch?v=qiUEgSCyY5o, 2022. Accedido: 8 de febrero de 2025.

[4] Omkar Hankare. Autoencoders explained. https://ompramod.medium.com/autoencoders-explained-1fa7f4c32f12, 2024. Accedido: 8 de febrero de 2025.

[5] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. https://arxiv.org/abs/1604.07379, 2016. Accedido: 8 de junio de 2025.

[6] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Z. Qureshi, and Mehran Ebrahimi. Edge-connect: Generative image inpainting with adversarial edge learning. https://arxiv.org/abs/1901.00212, 2019. Accedido: 13 de marzo de 2025.

[7] Andreas Lugmayr, Martin Danelljan, Andres Romero, Fisher Yu, Radu Timofte, and Luc Van Gool. Repaint: Inpainting using denoising diffusion probabilistic models. https://arxiv.org/abs/2201.09865, 2022. Accedido: 8 de junio de 2025.

[8] Chitwan Saharia, William Chan, Huiwen Chang, Chris A. Lee, Jonathan Ho, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models. https://arxiv.org/abs/2111.05826, 2022. Accedido: 8 de junio de 2025.

[9] Keras Team. Introduction to keras for engineers. https://keras.io/getting_started/intro_to_keras_for_engineers/, 2025. Accedido: 8 de junio de 2025.

[10] TensorFlow Team. Guía básica de tensorflow. https://www.tensorflow.org/guide/basics?hl=es, 2025. Accedido: 8 de junio de 2025.

[11] Vinod Nair and Geoffrey Hinton. Rectified linear units improve restricted boltzmann machines vinod nair. In *Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair*, volume 27, pages 807–814, 06 2010. Accedido: 12 de mayo de 2025.

[12] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. In *Rectifier Nonlinearities Improve Neural Network Acoustic Models*, 2013. Accedido: 28 de mayo de 2025.

[13] Janosh Riebesell, Stefan Bringuier, and Clemens Koza. Scientific Diagrams, 12 2022.

[14] Paul-Louis Pröve. An introduction to different types of convolutions in deep learning. https://medium.com/data-science/types-of-convolutions-in-deep-learning-717013397f4d, 07 2017. Accedido: 29 de mayo de 2025.

[15] O'Reilly Media. Learning tensorflow, chapter 4, convolutional neural networks. https://www.oreilly.com/library/view/learning-tensorflow/9781491978504/ch04.html. Accedido: 28 de mayo de 2025.

[16] David Díaz Solís. Blog: Cómo funciona el algoritmo de backpropagation. https://medium.com/@ddiazsolis/blog-cómo-funciona-el-algoritmo-de-backpropagation-22575308f14b, 2020. Accedido: 14 de mayo de 2025.

[17] Michael A. Nielsen. Chapter 2: How the backpropagation algorithm works. http://neuralnetworksanddeeplearning.com/chap2.html, 2015. Accedido: 29 de mayo de 2025.

[18] GeeksforGeeks. What is adam optimizer? https://www.geeksforgeeks.org/adam-optimizer/, 05 2025. Accedido: 17 de mayo de 2025.

[19] Keras Team. Adam optimizer. https://keras.io/api/optimizers/adam/, 05 2025.

[20] Donghwachaek. Upsampling. https://dacon.io/forum/406022, 02 2022. Accedido: 29 de mayo de 2025.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. https://arxiv.org/abs/1505.04597, 2015.

[22] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. https://arxiv.org/abs/1406.2661, 2014.

[23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. https://arxiv.org/abs/1706.03762, 2023.

[24] Stefania Cristina. The transformer attention mechanism. https://machinelearningmastery.com/the-transformer-attention-mechanism/, 01 2023. Accedido: 31 de mayo de 2025.

[25] Máxima. What is a transformer? https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04, 01 2019. Accedido: 31 de mayo de 2025.

[26] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. https://arxiv.org/abs/2010.11929, 2021.

[27] Skylar Jean Callis. Vision transformers, explained: A full walk-through of vision transformers. https://medium.com/data-science/vision-transformers-explained-a9d07147e4c8, 2024. Accedido: 31 de mayo de 2025.

[28] scikit-learn Developers. Model evaluation: Accuracy score. https://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score, 2025. Accedido: 1 de junio de 2025.

[29] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. https://www.cns.nyu.edu/pub/lcv/wang03-preprint.pdf, 2003. Accedido: 1 de junio de 2025.

[30] TensorFlow Team. tf.image.ssim — tensorflow. https://www.tensorflow.org/api_docs/python/tf/image/ssim, 2025. Accedido: 1 de junio de 2025.

[31] Lin Zhang, Lei Zhang, Xuanqin Mou, and David Zhang. Fsim: A feature similarity index for image quality assessment. *IEEE Transactions on Image Processing*, 20(8):2378–2386, 2011. Accedido: 8 de junio de 2025.

[32] Keyan Ding, Kede Ma, Shiqi Wang, and Eero P. Simoncelli. Image quality assessment: Unifying structure and texture similarity. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, page 1–1, 2020. Accedido: 8 de junio de 2025.

[33] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. https://arxiv.org/abs/1801.03924, 2018. Accedido: 8 de junio de 2025.

[34] GeeksforGeeks. Algorithms for image comparison. https://www.geeksforgeeks.org/algorithms-for-image-comparison/, Sep 2024. Accedido: 8 de junio de 2025.

[35] Wikipedia. Prueba de Shapiro-Wilk — Wikipedia, the free encyclopedia. http://es.wikipedia.org/w/index.php?title=Prueba%20de%20Shapiro-Wilk&oldid=164229225, 2025. Accedido: 8 de junio de 2025.

[36] DATAtab. Prueba de los rangos con signo de wilcoxon. https://datatab.es/tutorial/wilcoxon-test. Accedido: 12 de junio de 2025.

[37] Saul McLeod. What does effect size tell you? https://www.simplypsychology.org/effect-size.html. Accedido: 12 de junio de 2025.

[38] Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, July 1985. Accedido: 29 de abril de 2025.

[39] Raouf's Blog. Perlin noise: A procedural generation algorithm. https://rtouti.github.io/graphics/perlin-noise-algorithm, 2023. Accedido: 30 de mayo de 2025.

[40] ScratchAPixel. Perlin noise part 2: Terrain mesh. https://www.scratchapixel.com/lessons/procedural-generation-virtual-worlds/perlin-noise-part-2/perlin-noise-terrain-mesh.html. Accedido: 30 de mayo de 2025.

[41] Fan Yao and Yanli Chu. A generative image inpainting model based on edge and feature self-arrangement constraints. *Computational Intelligence and Neuroscience*, 2022(1):5904043, 2022. Accedido: 12 de marzo de 2025.

[42] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. https://arxiv.org/abs/1611.07004, 2018. Accedido: 18 de mayo de 2025.

[43] Historia Arte. Impresionismo. https://historia-arte.com/movimientos/impresionismo. Accedido: 10 de junio de 2025.

[44] Martín Prieto. 10 características del arte abstracto. https://artemartinprieto.com/2023/08/10-caracteristicas-del-arte-abstracto. Accedido: 10 de junio de 2025.

Apéndice A
Instalación

Esta sección enumera las librerías y sus versiones exactas usadas durante el desarrollo. Estas dependencias deben ser instaladas para ejecutar el *software* que acompaña a este trabajo. La versión de *Python* utilizada en todos los entornos definidos a continuación es *Python 3.11.11*.

Dependencias:

Entorno de entrenamiento del desarrollo del GAN:

1. Tensorflow 2.17.0
2. Keras 3.6.9
3. Numpy 1.26.4
4. Matplotlib 3.10.0

Interfaz e inferencia del modelo GAN:

1. Flask 3.1.0
2. Keras 3.6.0
3. Numpy 1.24.4
4. Opencv 4.10.0
5. Scikit-image 0.25.2
6. Scikit-learn 1.6.1
7. Torchvision 0.20.1
8. Piq 0.8.0

Interfaz e inferencia del modelo ICT:

1. Opencv 4.10.0

2. Scikit-image 0.25.2
3. Scikit-learn 1.6.1
4. Torchvision 0.20.1
5. Piq 0.8.0

Para entrenar el modelo o poner en marcha cualquiera de las interfaces, solo hay que estar en el directorio que contiene el archivo *GAN.py* o *interface.py* y ejecutarlo como script de **Python**.

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga