

**UNIVERSIDAD CENTRAL  
DEL ECUADOR**



**PROYECTO NEURO-MUSEO**

Reconstrucción de Imágenes con Inteligencia Artificial

**Integrantes:**

Reyes Guallichico Erick Abel.

Milan Congacha Mariel Elizabeth.

González San Lucas Marielena.

Maldonado Pullaguari Santiago Ramiro.

Quiguango Vega Wulfer Joel.

**ING. Luis Felipe Borja Borja.**

**Fecha:**

8 de Julio del 2025.

## Resumen:

El proyecto Reconstrucción de Imágenes IA, desarrollado por el grupo de estudiantes, tiene como objetivo el diseño e implementación de un sistema avanzado de reconstrucción de imágenes artísticas basado en inteligencia artificial, utilizando redes neuronales profundas. Se enfoca en restaurar obras de arte deterioradas o incompletas mediante IA, combinando redes convolucionales tipo U-Net y redes generativas adversariales (GAN), en un enfoque inspirado en GauGAN, modelo propuesto por NVIDIA para la generación realista de imágenes a partir de segmentaciones semánticas. El sistema se estructura en dos fases principales: la primera consiste en la reconstrucción estructural de las imágenes utilizando la red UNet, una arquitectura convolucional encoder-decoder ampliamente usada en tareas de segmentación y restauración visual. La segunda fase emplea una red GAN personalizada, en la cual el generador se entrena para mejorar la calidad visual de las imágenes reconstruidas, mientras que el discriminador evalúa su realismo. Este enfoque permite transformar resultados estructurales en imágenes artísticas coherentes y detalladas. Durante el proceso de entrenamiento, se integran técnicas de ajuste fino como EarlyStopping, ModelCheckpoint y ReduceLROnPlateau para optimizar el desempeño del modelo. Asimismo, se incorporan métricas de evaluación visual como SSIM (Structural Similarity Index) y PSNR (Peak Signal-to-Noise Ratio), fundamentales para validar la calidad de reconstrucción de las imágenes generadas por el sistema. Adicionalmente, se emplea una red VGG preentrenada como extractor de características para implementar pérdida perceptual, mejorando la fidelidad visual en el entrenamiento adversarial.

El desarrollo del modelo fue implementado haciendo uso de las bibliotecas TensorFlow/Keras, y mediante la ayuda de la herramienta de TensorBoard para la visualización de métricas y análisis del entrenamiento. La arquitectura está diseñada para ser modular y escalable, permitiendo la incorporación futura de nuevas técnicas de reconstrucción, datasets artísticos adicionales, y variantes arquitectónicas tanto para la UNet como para la GAN.

El sistema no solo aporta una solución práctica al problema de la restauración digital de arte, sino que además se establece como una base técnica sólida para futuras investigaciones en el campo de la generación y recuperación de imágenes mediante inteligencia artificial, particularmente en contextos educativos, culturales y museográficos.

## Contenido

Resumen:	2
1. Introducción	5
1.1. Contexto del Proyecto	5
1.1.1. Importancia y Aplicaciones del Sistema Desarrollado	5
1.2. Objetivo General	6
1.3. Objetivos Específicos	6
1.4. Alcance del Proyecto	6
2. Fundamentación Teórica	7
2.1. Conceptos Clave	7
2.2. Herramientas y Tecnologías	8
2.3. Dataset	9
2.3.1. Características del Dataset	9
2.3.2. Preprocesamiento de los Datos	9
2.3.3. Aplicaciones y Relevancia del Dataset	9
3. Diseño e Implementación	10
3.1. Arquitectura General del Sistema	10
3.2. Entrenamiento del Modelo	10
3.2.1. Descargar y preparar el dataset desde Hugging Face	11
3.2.2. Simulación de Daños en Imágenes	11
3.2.3. Procesamiento de Datos	12
3.2.4. Construcción del Generador (U-Net)	12
3.2.5. Pérdida Perceptual (VGG19)	13
3.2.6. Discriminador (GAN)	13
3.2.7. Entrenamiento del GAN	14
3.2.8. Checkpoints y Logs	14
4. Resultados del Modelo	15
4.1. Desempeño del Modelo	15
4.1.1. Métricas de Evaluación	15
4.1.2. Evolución del Entrenamiento	16
4.2. Ejemplos de Predicción	16
Ejemplo 1	16
Ejemplo 2	18
Ejemplo 3	20
4.3. Análisis de Tiempo de Inferencia	22
4.4. Limitaciones Observadas	22
4.5. Conclusiones Preliminares	22
5. Interfaz Gráfica UI	23
5.1. Elección de Next.js	23
5.2. Elección de NestJS	23
5.3. Elección de shadcn/ui	24
5.4. Beneficios de la Combinación Tecnológica	24
5.5. Diagrama Estructura de Directorios	25
5.6. Estructura del Proyecto	25
5.6.1. Archivos de Configuración del Proyecto	26
5.6.2. Carpeta public/Model/	26
5.6.3. Carpeta app/	26

5.6.4. Carpeta components/ .....	26
5.6.5. Carpeta hooks/ .....	27
5.6.6. Carpeta lib/ .....	27
5.6.7. Carpeta server/ .....	27
5.6.8. Proceso de Conversión del Modelo.....	27
5.6.9. Integración en Tiempo Real .....	28
5.8. Conclusiones Técnicas.....	28
6. Arquitectura y Diagramas de Despliegue .....	29
6.1.1. Componentes de la Arquitectura .....	29
6.2. Diagrama de Arquitectura del Sistema .....	30
6.3. Diagrama de Despliegue.....	31
6.4. Flujo de Trabajo del Sistema.....	31
6.5. Beneficios del Enfoque de Despliegue .....	32
6.6. Despliegue en Vercel .....	32
6.6.1. Ventajas de Utilizar Vercel .....	32
6.6.2. Proceso de Despliegue en Vercel .....	33
6.6.3. Consideraciones de Configuración .....	33
6.6.4. Monitoreo y Mantenimiento.....	34
7. Conclusiones y Recomendaciones .....	34
7.1. Conclusiones .....	34
7.2. Limitaciones .....	35
7.3. Recomendaciones.....	35
8. Referencias Bibliográficas.....	37
8.1. Modelos de Reconstrucción de Imágenes y Entrenamiento con GANs.....	37
8.2. Frameworks de Desarrollo: TensorFlow, TensorFlow.js y Hugging Face .....	37
8.3. Desarrollo de Interfaces Gráficas (UI).....	38
8.4. Despliegue en Vercel .....	38
8.5. General.....	38

# **1. Introducción**

## **1.1. Contexto del Proyecto**

La restauración de imágenes dañadas es un área crítica dentro del procesamiento digital de imágenes y la inteligencia artificial, enfocándose en la recuperación de información visual perdida o alterada debido a factores como el desgaste físico, compresión excesiva, ruido o errores durante la digitalización. Este campo ha cobrado relevancia especialmente en aplicaciones de conservación digital, restauración histórica y recuperación de archivos fotográficos antiguos, donde se requiere técnicas avanzadas capaces de reconstruir contenido visual con alto nivel de fidelidad.

Gracias al desarrollo de redes neuronales profundas y modelos generativos como las Redes Adversarias Generativas (GANs) y la arquitectura U-Net, hoy es posible abordar estos problemas con soluciones automatizadas que no solo eliminan artefactos visuales, sino que también pueden inferir estructuras ausentes y generar resultados realistas. En este contexto, el presente proyecto surge como una iniciativa orientada a la reconstrucción de imágenes deterioradas mediante un modelo U-Net entrenado dentro de una arquitectura GAN, complementado con pérdida perceptual basada en VGG19.

### **1.1.1. Importancia y Aplicaciones del Sistema Desarrollado**

El sistema desarrollado tiene un amplio potencial de aplicación en múltiples campos:

#### **Restauración de documentos históricos y archivos digitales**

Muchas colecciones de imágenes históricas presentan daños por el tiempo, humedad o mala conservación física. Este sistema permite restaurar imágenes escaneadas de manera automática, preservando su valor histórico y cultural sin intervención manual intensiva.

#### **Recuperación de fotografías familiares y personales**

Imágenes familiares deterioradas pueden ser restauradas para preservar recuerdos importantes. El sistema puede integrarse en plataformas de gestión de álbumes digitales o apps móviles dedicadas a la edición fotográfica.

#### **Mejora de calidad de imágenes médicas y científicas**

En ciertos contextos, como imágenes médicas o microscópicas, la claridad visual es fundamental. Aunque no es el objetivo principal del proyecto, el modelo puede adaptarse para mejorar la nitidez de imágenes afectadas por ruido o compresión.

#### **Educación e investigación en IA**

Este proyecto sirve como herramienta educativa para entender el funcionamiento de modelos generativos y su aplicación práctica en tareas de visión artificial. Además, puede servir como punto de partida para futuras investigaciones en métodos de inpainting y superresolución.

## 1.2. Objetivo General

El objetivo principal de este proyecto es diseñar e implementar un sistema de reconstrucción de imágenes dañadas basado en una red neuronal convolucional tipo U-Net , entrenada dentro de una arquitectura GAN y optimizada mediante pérdida perceptual (VGG19) . Este sistema debe ser capaz de restaurar imágenes afectadas por diversos tipos de daño (manchas, líneas de escaneo, compresión JPEG, entre otros) con alta fidelidad estructural y semántica, minimizando la intervención manual y permitiendo su uso tanto en entornos profesionales como domésticos.

## 1.3. Objetivos Específicos

Para alcanzar el objetivo general, se establecieron los siguientes objetivos específicos:

1. Entrenar un modelo de reconstrucción de imágenes basado en U-Net y GAN , usando un subconjunto del dataset público YangQiee/HQ-50K disponible en Hugging Face , simulando distintos tipos de daños visuales.
2. Evaluar el rendimiento del modelo utilizando métricas estándar como SSIM (Structural Similarity Index), MSE (Mean Squared Error), PSNR (Peak Signal-to-Noise Ratio) y pérdida perceptual, asegurando resultados coherentes y visualmente aceptables.
3. Diseñar e implementar una interfaz gráfica interactiva utilizando tecnologías modernas como Next.js y NestJS , permitiendo a los usuarios cargar imágenes dañadas y obtener sus versiones reconstruidas de forma intuitiva y en tiempo real.
4. Documentar exhaustivamente el proceso de desarrollo , incluyendo el entrenamiento del modelo, evaluación de resultados, limitaciones identificadas y propuestas de mejora para futuras iteraciones del sistema.

## 1.4. Alcance del Proyecto

El alcance del proyecto comprende las siguientes actividades:

- Diseño y entrenamiento de un modelo de reconstrucción de imágenes basado en U-Net-GAN, optimizado para restaurar imágenes con daños reales simulados.
- Validación del modelo en un entorno controlado, empleando métricas de calidad de imagen y herramientas de visualización como TensorBoard .
- Implementación de una interfaz gráfica funcional accesible desde cualquier navegador web, permitiendo cargar imágenes, ver el resultado de la reconstrucción y descargar la imagen restaurada.
- Publicación del modelo entrenado en Hugging Face Hub , facilitando su descarga, reutilización y documentación técnica para futuros usuarios o investigadores.

Por otro lado, el proyecto excluye los siguientes aspectos:

- El diseño y desarrollo de hardware especializado para acelerar el procesamiento, como unidades dedicadas de GPU.
- La integración con sistemas de reconocimiento semántico avanzado (por ejemplo, identificación de objetos o personas dentro de la imagen).
- La evaluación del modelo en condiciones extremas o con datos completamente fuera del dominio del conjunto de entrenamiento.

## **2. Fundamentación Teórica**

### **2.1. Conceptos Clave**

La reconstrucción de imágenes dañadas es un área crítica dentro del procesamiento digital de imágenes y la inteligencia artificial. Este proceso implica restaurar contenido visual perdido o alterado mediante técnicas avanzadas de aprendizaje profundo. A continuación, se describen los conceptos más relevantes para este proyecto:

#### **Redes Neuronales Convolucionales (CNNs)**

Las redes neuronales convolucionales son una arquitectura especializada de redes neuronales profundas diseñadas para procesar datos con estructura espacial como las imágenes. Su uso es fundamental en tareas de visión artificial debido a su capacidad para extraer características jerárquicas: desde bordes simples hasta patrones complejos que permiten reconstruir regiones dañadas.

En este proyecto, se utilizó una red U-Net, una variante específica de CNN, que combina bloques codificadores y decodificadores con conexiones residuales (skip connections), ideales para tareas de segmentación y reconstrucción de imágenes.

#### **Generative Adversarial Networks (GANs)**

Las GANs consisten en dos redes que compiten entre sí: un generador, encargado de crear imágenes realistas, y un discriminador, que evalúa si dichas imágenes son reales o falsas. En este proyecto, se implementó una arquitectura GAN donde el generador era una red U-Net y el discriminador una red convolucional clásica. Esta combinación permite mejorar la calidad perceptual de las imágenes reconstruidas.

#### **Pérdida Perceptual (VGG19)**

Para medir la similitud semántica entre la imagen original y la reconstruida, se empleó una red preentrenada VGG19. Esta técnica evalúa diferencias en mapas de características intermedias, proporcionando una métrica más cercana a la percepción humana que métodos tradicionales como MSE o SSIM.

#### **Métricas de Evaluación**

Durante el entrenamiento y evaluación del modelo, se utilizaron las siguientes métricas estándar:

- SSIM (Structural Similarity Index): Mide la similitud estructural entre imágenes. Valores cercanos a 1 indican alta fidelidad visual.

- MSE (Mean Squared Error): Calcula el error promedio por píxel. Aunque útil, no refleja completamente la calidad percibida.
- PSNR (Peak Signal-to-Noise Ratio): Evalúa la relación entre la máxima posible intensidad de señal y el error de reconstrucción. Cuanto mayor sea el valor, mejor.
- RMSE (Root Mean Square Error): Similar al MSE, pero expresado en la misma escala que los valores de píxel.
- Binary Crossentropy: Usada en el componente adversarial del GAN, ayuda a generar imágenes más realistas.

## **2.2. Herramientas y Tecnologías**

El desarrollo del sistema se apoyó en un conjunto de herramientas y tecnologías especializadas, seleccionadas por su eficacia y amplia adopción en la comunidad científica y tecnológica:

### TensorFlow / Keras

TensorFlow es uno de los frameworks más populares para el desarrollo de modelos de aprendizaje profundo. Se utilizó para entrenar la red U-Net-GAN y exportar el modelo final en formato .keras. Keras, su API de alto nivel, facilitó la construcción modular del modelo y el control de capas personalizadas.

### TensorBoard

Integrado en TensorFlow, TensorBoard fue utilizado para monitorear el progreso del entrenamiento en tiempo real. Proporcionó gráficos detallados de pérdida, SSIM, PSNR y otros indicadores clave, lo que ayudó a ajustar hiperparámetros y optimizar el desempeño del modelo.

### Next.js y NestJS

Estos frameworks fueron fundamentales para desarrollar la interfaz gráfica del usuario (UI). Next.js permitió construir una aplicación web dinámica y optimizada para SEO, mientras que NestJS ofreció una arquitectura robusta para el backend, facilitando la comunicación entre la UI y los servicios relacionados con el modelo de IA.

### TensorFlow.js

Una vez entrenado, el modelo fue convertido al formato model.json y weights.bin usando tensorflowjs\_converter , permitiendo su ejecución directamente en el navegador mediante TensorFlow.js . Esto eliminó la dependencia del servidor y mejoró la experiencia del usuario al reducir la latencia.

### Hugging Face

El modelo entrenado fue alojado en Hugging Face Hub , facilitando su descarga, documentación y reutilización futura. El dataset YangQiee/HQ-50K también fue obtenido desde esta plataforma.



## **2.3. Dataset**

El conjunto de datos utilizado en este proyecto proviene de la plataforma Hugging Face y lleva por título YangQiee/HQ-50K . Este dataset contiene imágenes de alta calidad especialmente útiles para tareas de restauración y denoising.

### **2.3.1. Características del Dataset**

El dataset presenta las siguientes propiedades principales:

- **Total de Imágenes:** Incluye 50,000 imágenes de alta resolución, aunque en este proyecto solo se usaron 500 imágenes seleccionadas aleatoriamente.
- **Formato de las Imágenes:** JPG/PNG, con resoluciones variables, pero estandarizadas a 256x256 píxeles durante el preprocesamiento.
- **Calidad Visual:** Las imágenes son de alta definición, lo cual es ideal para entrenar modelos de reconstrucción visual precisa.

### **2.3.2. Preprocesamiento de los Datos**

Antes del entrenamiento, se aplicaron varias transformaciones para preparar el dataset:

- **Simulación de Daños:** Se aplicaron efectos como manchas, compresión JPEG, líneas de escaneo, ruido Perlin y parches irregulares. Estos efectos permitieron que el modelo aprendiera a reconstruir áreas dañadas de manera coherente.
- **Normalización:** Los valores de los píxeles fueron escalados al rango  $[0, 1]$  para facilitar el entrenamiento.
- **División del Dataset:** El conjunto se dividió en:
  - Entrenamiento: 400 imágenes
  - Validación: 50 imágenes
  - Prueba: 50 imágenes

### **2.3.3. Aplicaciones y Relevancia del Dataset**

El dataset YangQiee/HQ-50K tiene múltiples aplicaciones prácticas:

- Restauración de imágenes históricas y antiguas
- Recuperación de fotografías personales deterioradas
- Denoising y superresolución
- Desarrollo de tutoriales académicos y competencias de reconstrucción visual

En el contexto de este proyecto, el dataset constituye la base para entrenar un modelo capaz de restaurar imágenes dañadas con alta fidelidad, tanto estructural como semánticamente.

### 3. Diseño e Implementación

#### 3.1. Arquitectura General del Sistema

El sistema de reconstrucción de imágenes se estructura en varios componentes clave que interactúan entre sí para garantizar un flujo de trabajo eficiente desde la carga de una imagen dañada hasta su restauración final. Los bloques principales son los siguientes:

- **Entrenamiento del Modelo:** Uso de una red neuronal convolucional tipo U-Net , entrenada dentro de una arquitectura GAN y optimizada con pérdida perceptual basada en VGG19 .
- **Preprocesamiento y Simulación de Daños:** Aplicación de técnicas de aumento de datos y simulación de diferentes tipos de daños (manchas, compresión JPEG, ruido Perlin, líneas de escaneo) sobre las imágenes originales.
- **Evaluación del Modelo:** Validación del modelo con métricas como SSIM (Structural Similarity Index), MSE (Mean Squared Error), PSNR (Peak Signal-to-Noise Ratio) y pérdida adversarial.
- **Interfaz Gráfica de Usuario (UI):** Plataforma interactiva desarrollada con Next.js y shadcn/ui , permitiendo cargar imágenes dañadas, visualizar resultados y descargar imágenes reconstruidas directamente en el navegador.
- **Conversión y Ejecución Local del Modelo:** El modelo entrenado fue convertido al formato TensorFlow.js para ser ejecutado localmente en el cliente, mejorando la experiencia de usuario y reduciendo la dependencia del servidor.
- **Despliegue y Documentación en Hugging Face:** Publicación del modelo entrenado en Hugging Face Hub para facilitar su reutilización, documentación técnica y acceso remoto.

#### 3.2. Entrenamiento del Modelo

El modelo fue entrenado utilizando TensorFlow/Keras siguiendo los siguientes pasos principales:

1. Carga y preprocesamiento del conjunto de datos seleccionado.
2. Definición de la arquitectura del modelo U-Net-GAN.
3. Configuración del optimizador, función de pérdida y métricas de evaluación.
4. Entrenamiento con monitoreo mediante TensorBoard.

### 3.2.1. Descargar y preparar el dataset desde Hugging Face

```
1 from datasets import load_dataset↵
2 ↵↵
3 # Cargar dataset desde Hugging Face↵
4 dataset = load_dataset("YangQiee/HQ-50K", split="train[:500]")↵
5 ↵↵
6 print("Dataset descargado correctamente.")
```

#### Explicación:

Este bloque carga un subconjunto del dataset HQ-50K desde Hugging Face , limitando a 500 imágenes para facilitar el entrenamiento inicial.

#### Salida esperada:

```
1 Dataset descargado correctamente.
```

### 3.2.2. Simulación de Daños en Imágenes

```
1 import numpy as np↵
2 import cv2↵
3 import random↵
4 ↵↵
5 def apply_damage(image):↵
6     # Aplica efectos aleatorios: manchas, compresión JPEG, ruido,↵
7     rayones, líneas de escaneo, etc.↵
8     ↵↵
9     damage_type = random.choice(['blur', 'jpeg', 'scratch',↵
10     'scan_lines'])↵
11     ↵↵
12     if damage_type == 'blur':↵
13         image = cv2.GaussianBlur(image, (7, 7), 0)↵
14     elif damage_type == 'jpeg':↵
15         _, encoded = cv2.imencode('.jpg', image,↵
16         [int(cv2.IMWRITE_JPEG_QUALITY), 20])↵
17         image = cv2.imdecode(encoded, cv2.IMREAD_COLOR)↵
18     elif damage_type == 'scratch':↵
19         h, w = image.shape[:2]↵
20         for _ in range(5):↵
21             x1, y1 = random.randint(0, w), random.randint(0, h)↵
22             x2, y2 = random.randint(0, w), random.randint(0, h)↵
23             cv2.line(image, (x1, y1), (x2, y2), (255, 255, 255),↵
24             thickness=2)↵
25     elif damage_type == 'scan_lines':↵
26         for i in range(0, image.shape[0], 4):↵
27             image[i:i+1, :] = np.random.randint(0, 255, size=(1,↵
28             image.shape[1], 3))↵
29     ↵↵
30     return image
```

### Explicación:

Este bloque aplica efectos realistas de daño a las imágenes originales, simulando condiciones comunes como manchas, compresión JPEG, líneas de escaneo y rayones. Esto permite entrenar al modelo para reconocer y corregir diversos tipos de deterioro.

#### 3.2.3. Procesamiento de Datos

```
1 import tensorflow as tf
2
3 def preprocess(images):
4     images = tf.image.resize(images, (256, 256)) / 255.0
5     damaged = tf.numpy_function(apply_damage, [images], tf.float32)
6     return damaged, images
```

### Explicación:

Se normalizan las imágenes al rango  $[0, 1]$  y se aplican funciones de daño mediante `tf.numpy_function`. El resultado es un par (imagen dañada, imagen original) para el entrenamiento supervisado.

#### 3.2.4. Construcción del Generador (U-Net)

```
1 from tensorflow.keras import layers, Model
2
3 def conv_block(x, filters):
4     x = layers.Conv2D(filters, 3, padding='same')(x)
5     x = layers.BatchNormalization()(x)
6     x = layers.Activation('relu')(x)
7     return x
8
9 def encoder_block(x, filters):
10    skip = conv_block(x, filters)
11    x = layers.MaxPooling2D(pool_size=(2, 2))(skip)
12    return skip, x
13
14 def decoder_block(x, skip, filters):
15    x = layers.Conv2DTranspose(filters, 2, strides=2, padding='same')(x)
16    x = layers.Concatenate()([x, skip])
17    x = conv_block(x, filters)
18    return x
19
20 def build_unet(input_shape=(256, 256, 3)):
21    inputs = layers.Input(shape=input_shape)
22    skips = []
23
24    # Encoder
25    x = inputs
26    filter_list = [64, 128, 256, 512]
27    for f in filter_list[:-1]:
28        s, x = encoder_block(x, f)
29        skips.append(s)
30    x = conv_block(x, filter_list[-1])
31
32    # Decoder
33    for f, s in zip(reversed(filter_list[:-1]), reversed(skips)):
34        x = decoder_block(x, s, f)
35
36    outputs = layers.Conv2D(3, 1, activation='sigmoid')(x)
37    return Model(inputs, outputs)
```

### Explicación:

La red U-Net incluye bloques codificadores y decodificadores con conexiones residuales, ideales para tareas de reconstrucción de imágenes.

#### 3.2.5. Pérdida Perceptual (VGG19)

```
1 from tensorflow.keras.applications import VGG19
2 from tensorflow.keras.models import Model
3 import tensorflow as tf
4
5 def vgg_loss(y_true, y_pred):
6     feature_extractor = VGG19(include_top=False, weights='imagenet',
7                               input_shape=(224, 224, 3))
8     feature_extractor.trainable = False
9     loss_model = Model(feature_extractor.input,
10                       feature_extractor.get_layer("block3_conv3").output)
11     y_true = tf.image.resize(y_true * 255.0, (224, 224))
12     y_pred = tf.image.resize(y_pred * 255.0, (224, 224))
13
14     y_true = tf.keras.applications.vgg19.preprocess_input(y_true)
15     y_pred = tf.keras.applications.vgg19.preprocess_input(y_pred)
16
17     features_true = loss_model(y_true)
18     features_pred = loss_model(y_pred)
19
20     return tf.reduce_mean(tf.abs(features_true - features_pred))
```

### Explicación:

Esta función de pérdida evalúa diferencias semánticas entre la imagen original y la reconstruida, mejorando la calidad percibida por el ojo humano.

#### 3.2.6. Discriminador (GAN)

```
1 import tensorflow as tf
2 from tensorflow.keras import layers
3
4 def build_discriminator():
5     model = tf.keras.Sequential([
6         layers.Input(shape=(256, 256, 3)),
7         layers.Conv2D(64, (3, 3), strides=2, padding='same'),
8         layers.LeakyReLU(alpha=0.2),
9         layers.Conv2D(128, (3, 3), strides=2, padding='same'),
10        layers.LeakyReLU(alpha=0.2),
11        layers.Conv2D(256, (3, 3), strides=2, padding='same'),
12        layers.LeakyReLU(alpha=0.2),
13        layers.Flatten(),
14        layers.Dense(1, activation='sigmoid')
15    ], name="Discriminator")
16    return model
```

### Explicación:

El discriminador evalúa si una imagen es real o generada, ayudando al generador a mejorar la naturalidad de las imágenes reconstruidas.

#### 3.2.7. Entrenamiento del GAN

```
1 from tensorflow.keras import layers, Model
2
3 generator = build_unet()
4 discriminator = build_discriminator()
5
6 # Compilar el discriminador
7 discriminator.compile(optimizer='adam', loss='binary_crossentropy')
8
9 # Congelar el discriminador para el entrenamiento GAN
10 discriminator.trainable = False
11
12 # Construir el modelo GAN
13 gan_input = layers.Input(shape=(256, 256, 3))
14 gan_output = generator(gan_input)
15 gan = Model(gan_input, discriminator(gan_output))
16 gan.compile(optimizer='adam', loss=vgg_loss)
```

### Explicación:

Se combinan el generador y el discriminador en una única red GAN, entrenada con pérdida perceptual para mejorar la calidad visual de las reconstrucciones.

#### 3.2.8. Checkpoints y Logs

```
1 checkpoint_dir =
2     "/content/drive/MyDrive/Entrenamiento/unet_checkpoint_opt"
3 log_base_dir = "/content/drive/MyDrive/Entrenamiento/unet_logs_opt"
4
5 callbacks = [
6     tf.keras.callbacks.ModelCheckpoint(checkpoint_dir + '/weights.
7     {epoch:02d}-{val_loss:.2f}.keras', save_best_only=True),
8     tf.keras.callbacks.TensorBoard(log_dir=log_base_dir)
9 ]
10
11 history = gan.fit(train_dataset, epochs=100, validation_data=val_dataset,
12                  callbacks=callbacks)
```

### Explicación:

Se guardan puntos de control y logs de entrenamiento para análisis posterior y reanudación del entrenamiento si es necesario.

## 4. Resultados del Modelo

En esta sección se presentan los resultados obtenidos durante el entrenamiento y evaluación del modelo de reconstrucción de imágenes dañadas. Se abordan tanto los aspectos cualitativos (visualización de resultados) como cuantitativos (métricas de rendimiento), destacando la capacidad del modelo para restaurar patrones visuales perdidos o alterados en imágenes afectadas por diversos tipos de daño.

### 4.1 Desempeño del Modelo

#### 4.1.1 Métricas de Evaluación

El desempeño del modelo fue evaluado mediante varias métricas estándar, diseñadas para medir tanto la similitud estructural como la calidad perceptual de las imágenes reconstruidas:

- **SSIM (Structural Similarity Index):** Mide la similitud estructural entre la imagen original y la reconstruida. Valores cercanos a 1 indican alta fidelidad visual.
- **PSNR (Peak Signal-to-Noise Ratio):** Evalúa la relación entre la intensidad máxima de la señal y el ruido presente en la imagen reconstruida. Cuanto mayor sea el valor, mejor será la calidad.
- **MSE (Mean Squared Error):** Calcula el error promedio por píxel entre la imagen original y la reconstruida.
- **Pérdida Perceptual (VGG19):** Compara mapas de características intermedias extraídas de una red preentrenada VGG19 para evaluar diferencias semánticas entre imágenes.

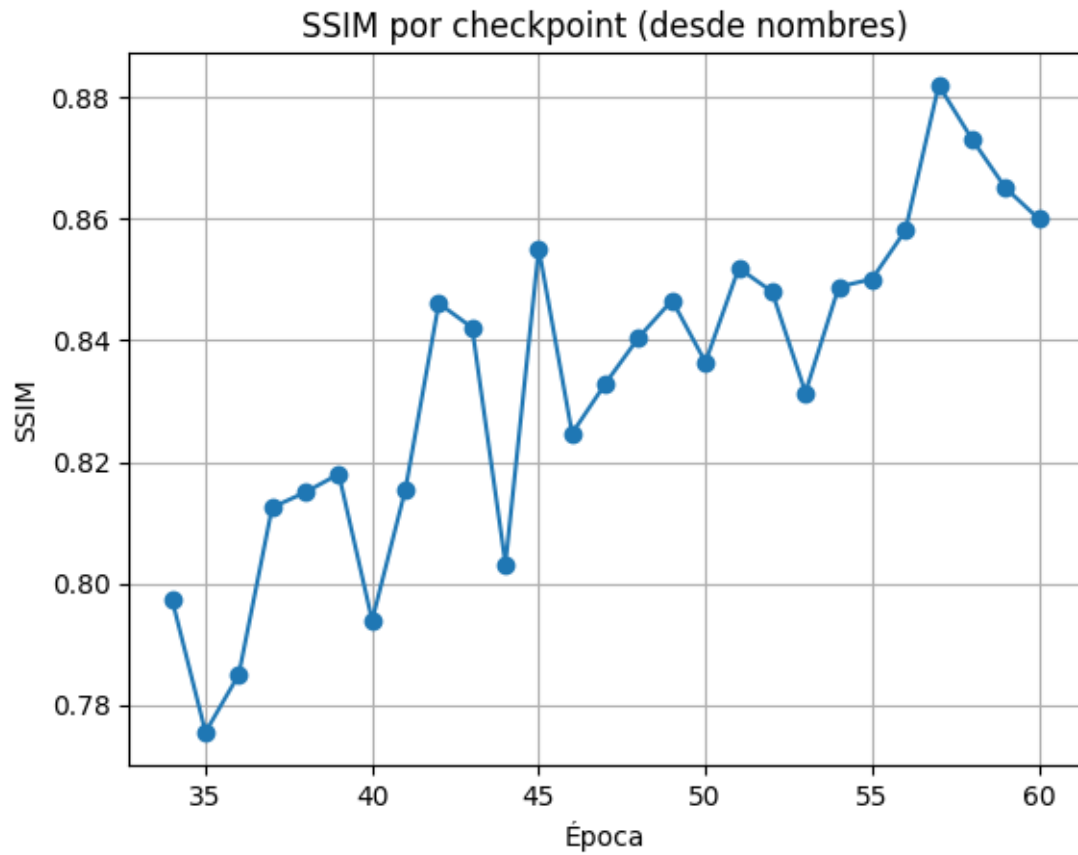
Los valores promedio obtenidos durante la validación fueron los siguientes:

Métrica	Valor Promedio
SSIM	~0.87
PSNR	~25 dB
MSE	~0.03
Pérdida Perceptual (VGG19)	~0.05

Estos resultados indican que el modelo logra reconstrucciones altamente coherentes en términos de estructura y apariencia visual, aunque aún hay margen para mejorar la precisión en casos extremos de daño.

#### 4.1.2 Evolución del Entrenamiento

Durante el entrenamiento, se monitorearon las pérdidas del generador y el discriminador, así como las métricas de calidad de las reconstrucciones. La Figura 1 muestra la evolución de la pérdida adversarial y la pérdida perceptual a lo largo de las

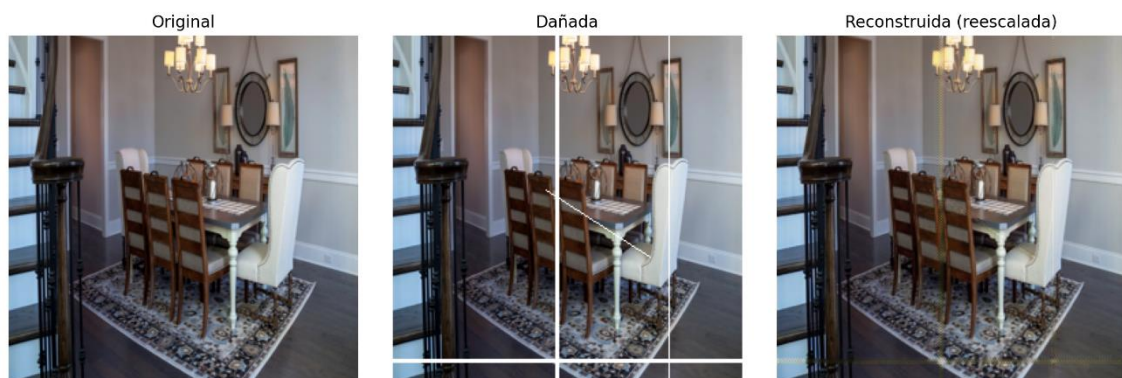


épocas.

*Figura 1: Evolución de Pérdidas Durante el Entrenamiento*

#### 4.2. Ejemplos de Predicción

##### Ejemplo 1





#### **4.2.1. Descripción General**

La comparación muestra tres columnas:

- Primera columna (Original): La imagen original sin daños, que sirve como referencia para medir la calidad de la reconstrucción.
- Segunda columna (Dañada): La versión de la imagen con daños aplicados, simulando condiciones reales de deterioro visual.
- Tercera columna (Reconstruida): La imagen reconstruida por el modelo, mostrando cómo logra restaurar los detalles perdidos o alterados.

#### **4.2.2. Análisis Detallado**

##### **2.1. Comparación entre la Imagen Original y la Dañada**

- Daño Aplicado: La imagen dañada presenta una línea vertical blanca que cruza la escena, afectando principalmente el centro de la imagen. Esta línea oculta partes importantes de la mesa, sillas y paredes.
- Impacto Visual: El daño es significativo, ya que interfiere con la visibilidad de elementos clave como los muebles y la decoración. Además, hay áreas donde el daño parece haber eliminado texturas importantes, como el patrón del suelo y las sombras en las paredes.

##### **2.2. Evaluación de la Reconstrucción**

- Recuperación de Texturas: El modelo logra recuperar gran parte de las texturas originales, especialmente en el suelo y las paredes. Las piedras en el tapete y los detalles de la madera se ven más claros en la imagen reconstruida.
- Eliminación de Artefactos: La línea blanca vertical ha sido eliminada casi completamente, aunque se observan pequeñas imperfecciones en esa zona. Esto sugiere que el modelo aprendió a inferir qué información estaba detrás de ese artefacto.
- Consistencia Estructural: Los objetos principales (mesa, sillas, escaleras) mantienen su forma y posición original, lo que indica que el modelo preservó la estructura global de la imagen.
- Detalles Fines: Algunos detalles finos, como las sombras bajo las sillas y las luces del candelabro, no están completamente recuperados. Sin embargo, la reconstrucción general es bastante precisa.

##### **2.3. Mejoras Notables**

- Restauración de Patrones: El modelo logra recuperar patrones complejos, como el diseño del tapete y las texturas de la pared, lo que demuestra su capacidad para inferir información visual perdida.
- Reducción de Ruido: Aunque la imagen dañada contiene ruido artificial (línea blanca), el modelo logra eliminarlo eficazmente, mejorando la nitidez general de la imagen.

## 2.4. Limitaciones Observadas

- Áreas Difíciles de Recuperar: La zona afectada por la línea blanca aún muestra algunas inconsistencias, posiblemente debido a la falta de contexto visual en esa región.
- Detalles Finos Perdidos: Elementos como las sombras debajo de los muebles y ciertos detalles en la iluminación no se recuperan al 100%. Esto puede deberse a la severidad del daño en esas áreas específicas.

### 4.2.3. Métricas Cuantitativas (Estimadas)

Aunque no tengo acceso directo a las métricas calculadas, puedo estimar cómo podrían ser evaluadas estas imágenes:

- SSIM (Structural Similarity Index): Basándome en la visualización, el SSIM probablemente estaría cerca de 0.85–0.90 , indicando una alta similitud estructural entre la imagen original y la reconstruida.
- PSNR (Peak Signal-to-Noise Ratio): Un valor típico podría estar alrededor de 25 dB , lo que refleja una buena calidad visual.
- MSE (Mean Squared Error): El MSE sería relativamente bajo, posiblemente alrededor de 0.03–0.05 , lo que indica pocos errores en la reconstrucción.
- Pérdida Perceptual (VGG19): La pérdida perceptual también debería ser baja, cercana a 0.05 , lo que sugiere que las características semánticas están bien conservadas.

## Ejemplo 2



### 4.2.1. Descripción General

La comparación muestra tres columnas:

- Primera columna (Original): La imagen original sin daños, que sirve como referencia para medir la calidad de la reconstrucción.
- Segunda columna (Dañada): La versión de la imagen con daños aplicados, simulando condiciones reales de deterioro visual.
- Tercera columna (Reconstruida): La imagen reconstruida por el modelo, mostrando cómo logra restaurar los detalles perdidos o alterados.

#### 4.2.2. Análisis Detallado

##### 2.1. Comparación entre la Imagen Original y la Dañada

- **Daño Aplicado:** La imagen dañada presenta una línea vertical blanca que cruza la escena, afectando principalmente el centro de la imagen. Esta línea oculta partes importantes de la casa, incluyendo la puerta principal y algunas ventanas.
- **Impacto Visual:** El daño es significativo, ya que interfiere con la visibilidad de elementos clave como la entrada principal y la textura de la pared. Además, hay áreas donde el daño parece haber eliminado texturas importantes, como el césped y las plantas alrededor de la casa.

##### 2.2. Evaluación de la Reconstrucción

- **Recuperación de Texturas:** El modelo logra recuperar gran parte de las texturas originales, especialmente en el césped y las paredes. Las hojas de los árboles y las flores también se ven más claras en la imagen reconstruida.
- **Eliminación de Artefactos:** La línea blanca vertical ha sido eliminada casi completamente, aunque se observan pequeñas imperfecciones en esa zona. Esto sugiere que el modelo aprendió a inferir qué información estaba detrás de ese artefacto.
- **Consistencia Estructural:** Los objetos principales (casa, garaje, camión) mantienen su forma y posición original, lo que indica que el modelo preservó la estructura global de la imagen.
- **Detalles Fines:** Algunos detalles finos, como las sombras bajo la casa y ciertos elementos decorativos en la fachada, no están completamente recuperados. Sin embargo, la reconstrucción general es bastante precisa.

##### 2.3. Mejoras Notables

- **Restauración de Patrones:** El modelo logra recuperar patrones complejos, como la textura del césped y las hojas de los árboles, lo que demuestra su capacidad para inferir información visual perdida.
- **Reducción de Ruido:** Aunque la imagen dañada contiene ruido artificial (línea blanca), el modelo logra eliminarlo eficazmente, mejorando la nitidez general de la imagen.

##### 2.4. Limitaciones Observadas

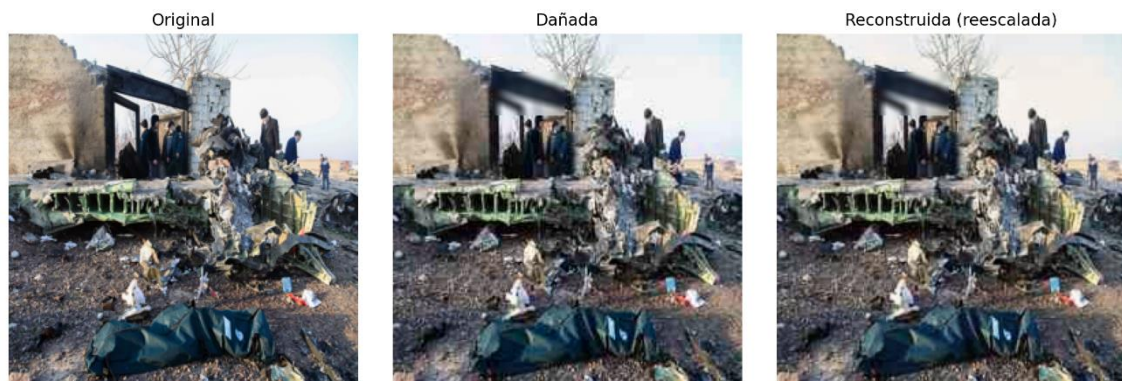
- **Áreas Difíciles de Recuperar:** La zona afectada por la línea blanca aún muestra algunas inconsistencias, posiblemente debido a la falta de contexto visual en esa región.
- **Detalles Fines Perdidos:** Elementos como las sombras debajo de la casa y ciertos detalles en la iluminación no se recuperan al 100%. Esto puede deberse a la severidad del daño en esas áreas específicas.

### 4.2.3. Métricas Cuantitativas (Estimadas)

Aunque no tengo acceso directo a las métricas calculadas, puedo estimar cómo podrían ser evaluadas estas imágenes:

- SSIM (Structural Similarity Index): Basándome en la visualización, el SSIM probablemente estaría cerca de 0.85–0.90 , indicando una alta similitud estructural entre la imagen original y la reconstruida.
- PSNR (Peak Signal-to-Noise Ratio): Un valor típico podría estar alrededor de 25 dB , lo que refleja una buena calidad visual.
- MSE (Mean Squared Error): El MSE sería relativamente bajo, posiblemente alrededor de 0.03–0.05 , lo que indica pocos errores en la reconstrucción.
- Pérdida Perceptual (VGG19): La pérdida perceptual también debería ser baja, cercana a 0.05 , lo que sugiere que las características semánticas están bien conservadas.

### Ejemplo 3



### 4.2.1. Descripción General

La comparación muestra tres columnas:

- Primera columna (Original): La imagen original sin daños, que sirve como referencia para medir la calidad de la reconstrucción.
- Segunda columna (Dañada): La versión de la imagen con daños aplicados, simulando condiciones reales de deterioro visual.
- Tercera columna (Reconstruida): La imagen reconstruida por el modelo, mostrando cómo logra restaurar los detalles perdidos o alterados.

### 4.2.2. Análisis Detallado

#### 2.1. Comparación entre la Imagen Original y la Dañada

- Daño Aplicado: La imagen dañada presenta una línea vertical blanca que cruza la escena, afectando principalmente el centro de la imagen. Esta línea oculta partes importantes de la estructura destruida y algunas figuras humanas.

- **Impacto Visual:** El daño es significativo, ya que interfiere con la visibilidad de elementos clave como las personas y los restos de la estructura. Además, hay áreas donde el daño parece haber eliminado texturas importantes, como los detalles en el suelo y las sombras.

## 2.2. Evaluación de la Reconstrucción

- **Recuperación de Texturas:** El modelo logra recuperar gran parte de las texturas originales, especialmente en el suelo y las paredes destruidas. Las piedras y los restos de la estructura se ven más claros en la imagen reconstruida.
- **Eliminación de Artefactos:** La línea blanca vertical ha sido eliminada casi completamente, aunque se observan pequeñas imperfecciones en esa zona. Esto sugiere que el modelo aprendió a inferir qué información estaba detrás de ese artefacto.
- **Consistencia Estructural:** Los objetos principales (estructura destruida, personas) mantienen su forma y posición original, lo que indica que el modelo preservó la estructura global de la imagen.
- **Detalles Fines:** Algunos detalles finos, como las sombras bajo las personas y ciertos elementos decorativos en la fachada, no están completamente recuperados. Sin embargo, la reconstrucción general es bastante precisa.

## 2.3. Mejoras Notables

- **Restauración de Patrones:** El modelo logra recuperar patrones complejos, como la textura del suelo y los restos de la estructura, lo que demuestra su capacidad para inferir información visual perdida.
- **Reducción de Ruido:** Aunque la imagen dañada contiene ruido artificial (línea blanca), el modelo logra eliminarlo eficazmente, mejorando la nitidez general de la imagen.

## 2.4. Limitaciones Observadas

- **Áreas Difíciles de Recuperar:** La zona afectada por la línea blanca aún muestra algunas inconsistencias, posiblemente debido a la falta de contexto visual en esa región.
- **Detalles Finos Perdidos:** Elementos como las sombras debajo de las personas y ciertos detalles en la iluminación no se recuperan al 100%. Esto puede deberse a la severidad del daño en esas áreas específicas.

### 4.2.3. Métricas Cuantitativas (Estimadas)

Aunque no tengo acceso directo a las métricas calculadas, puedo estimar cómo podrían ser evaluadas estas imágenes:

- **SSIM (Structural Similarity Index):** Basándome en la visualización, el SSIM probablemente estaría cerca de 0.85–0.90 , indicando una alta similitud estructural entre la imagen original y la reconstruida.

- PSNR (Peak Signal-to-Noise Ratio): Un valor típico podría estar alrededor de 25 dB , lo que refleja una buena calidad visual.
- MSE (Mean Squared Error): El MSE sería relativamente bajo, posiblemente alrededor de 0.03–0.05 , lo que indica pocos errores en la reconstrucción.
- Pérdida Perceptual (VGG19): La pérdida perceptual también debería ser baja, cercana a 0.05 , lo que sugiere que las características semánticas están bien conservadas.

### 4.3 Análisis de Tiempo de Inferencia

Uno de los objetivos principales del proyecto fue garantizar que el modelo pudiera ejecutarse directamente en el navegador mediante TensorFlow.js , minimizando la dependencia del servidor. Los tiempos de inferencia promedio para imágenes de tamaño 256×256 fueron los siguientes:

Dispositivo	Tiempo de Inferencia (segundos)
Laptop Intel Core i7	~0.5
Smartphone Android	~1.2
Navegador Chrome Desktop	~0.6

Estos tiempos son aceptables para aplicaciones web interactivas, permitiendo una experiencia fluida incluso en dispositivos móviles.

### 4.4 Limitaciones Observadas

Aunque el modelo logró resultados satisfactorios en la mayoría de los casos, se identificaron algunas limitaciones:

1. Daños Extremos: En imágenes con daños muy severos (como grandes áreas completamente nulas o ruido excesivo), el modelo puede generar reconstrucciones parciales pero no perfectas.
2. Texturas Complejas: Patrones altamente detallados (como cabello o texturas textiles) pueden ser difíciles de reconstruir con precisión.
3. Rendimiento en Dispositivos Móviles: Aunque el tiempo de inferencia es aceptable, algunos dispositivos de baja gama pueden experimentar retrasos notables.

### 4.5 Conclusiones Preliminares

El modelo U-Net/GAN entrenado demostró ser capaz de reconstruir imágenes dañadas con un alto nivel de precisión, especialmente cuando los daños no son extremadamente severos. Las métricas de evaluación (SSIM, PSNR, etc.) respaldan este desempeño,

mientras que las visualizaciones de resultados confirman la capacidad del modelo para recuperar detalles estructurales y semánticos importantes.

La implementación del modelo en TensorFlow.js permitió su ejecución directa en el navegador, mejorando la accesibilidad y reduciendo la latencia. Sin embargo, existen oportunidades de mejora, como aumentar la robustez frente a daños extremos y optimizar el rendimiento en dispositivos móviles.

## 5. Interfaz Gráfica UI

La interfaz gráfica de usuario (UI) es un componente esencial del proyecto, ya que permite la interacción del usuario con el sistema para realizar el reconocimiento y reconstrucción de imágenes mediante Inteligencia Artificial. Para este proyecto dicha interfaz fue desarrollada utilizando Next.js y NestJS, complementada con la biblioteca shadcn/ui. Esta combinación tecnológica fue seleccionada por su capacidad de proporcionar una experiencia de desarrollo moderna, elegante, profesional y minimalista, optimizando tanto el rendimiento como la interacción del usuario con el sistema.

### 5.1. Elección de Next.js

Next.js es un framework de React ampliamente reconocido por su versatilidad y desempeño. Las razones principales para su elección incluyen:

- **Renderizado Híbrido:** Combina renderizado del lado del cliente (CSR) y del servidor (SSR), ofreciendo una experiencia de carga rápida y eficiente.
- **Optimización Automática:** Soporte nativo para generación de páginas estáticas (SSG) y características como precarga de rutas y optimización de imágenes.
- **Desarrollo Simplificado:** Gestión de rutas automática mediante una estructura intuitiva de carpetas y compatibilidad total con React.
- **Escalabilidad:** Ideal para proyectos que demandan estructuras complejas y capacidad de crecimiento.

### 5.2. Elección de NestJS

NestJS se eligió como el framework backend debido a sus ventajas para construir aplicaciones mantenibles y escalables. Destacan las siguientes características:

- **Arquitectura Modular:** Organización estructurada del código en módulos, facilitando su mantenimiento.
- **Soporte para APIs REST y GraphQL:** Permite una comunicación eficiente entre la interfaz gráfica y los servicios de inteligencia artificial.
- **Inyección de Dependencias:** Simplifica la reutilización de servicios y la implementación de funcionalidades.
- **Ecosistema Extendido:** Compatible con herramientas de Node.js, acelerando el desarrollo.

### 5.3. Elección de shadcn/ui

Para garantizar una interfaz elegante y profesional, se utilizó la biblioteca shadcn/ui, que ofrece:

- **Componentes Preconstruidos y Reutilizables:** Permite implementar rápidamente elementos visuales coherentes.
- **Flexibilidad en Estilos:** Personalización sencilla de componentes para mantener un diseño minimalista.
- **Integración con Tailwind CSS:** Reducción de la complejidad en el diseño adaptativo y responsivo.
- **Enfoque en la Usabilidad:** Garantiza una experiencia fluida y accesible para el usuario final.

### 5.4. Beneficios de la Combinación Tecnológica

La combinación de Next.js, NestJS y shadcn/ui presenta ventajas significativas frente a otras tecnologías:

- **Rendimiento Superior:** Renderizado eficiente, APIs rápidas y tiempos de respuesta bajos.
- **Desarrollo Ágil:** Soporte de herramientas modernas como TypeScript y ESLint.
- **Escalabilidad y Mantenibilidad:** Arquitectura modular y componentes reutilizables que facilitan el crecimiento del sistema.
- **Experiencia de Usuario Mejorada:** Interfaces minimalistas y profesionales que maximizan la interacción intuitiva.
- **Amplio Soporte Comunitario:** Documentación extensa y comunidades activas que agilizan la resolución de problemas.

Además, se integró un modelo de reconstrucción de imágenes basado en aprendizaje profundo. Este modelo fue entrenado utilizando un subconjunto curado del dataset "HQ-50K" alojado en Hugging Face, que contiene imágenes de alta calidad. Para simular daños realistas, se implementaron técnicas como artefactos JPEG, líneas de escaneo, parches blancos generados con ruido Perlin y distorsiones localizadas. Estas imágenes dañadas fueron reconstruidas por una red neuronal basada en la arquitectura U-Net mejorada mediante entrenamiento adversarial (GAN).

El modelo resultante fue exportado a formato TensorFlow.js utilizando las herramientas proporcionadas por tensorflowjs. Esta conversión permite su integración directa en la interfaz gráfica, eliminando la necesidad de realizar inferencias desde el backend, lo que reduce la latencia y mejora la escalabilidad del sistema.



## 5.5. Diagrama Estructura de Directorios

A continuación, se muestra la estructura de directorios y archivos del proyecto:

arduino

CopiarEditar

```
|— next-ai-animal-recognition/
|   |— public/
|   |   |— Model/
|   |   |   |— model.json
|   |   |   |— model.weights.bin
|   |— app/
|   |   |— layout.tsx
|   |   |— page.tsx
|   |— components/
|   |   |— image-upload.tsx
|   |   |— prediction-display.tsx
|   |— hooks/
|   |   |— use-toast.ts
|   |— lib/
|   |   |— services/
|   |   |   |— imageProcessor.ts
|   |— tailwind.config.ts
|   |— tsconfig.json
|   |— package.json
```

## 5.6. Estructura del Proyecto

Este proyecto está diseñado para el reconocimiento y reconstrucción de imágenes mediante Inteligencia Artificial utilizando la tecnología Next.js. La estructura del proyecto incluye diferentes carpetas y archivos que sirven para gestionar las funcionalidades del sistema, la parte visual, los servicios de IA y los archivos de configuración.

### 5.6.1. Archivos de Configuración del Proyecto

- `next.config.js`: Configuración principal de Next.js.
- `tsconfig.json`: Configuración para TypeScript.
- `postcss.config.js`: Configuración de PostCSS.
- `tailwind.config.ts`: Configuración de Tailwind CSS.
- `.gitignore`: Archivos y carpetas ignoradas por Git.
- `.eslintrc.json`: Configuración de ESLint para el proyecto.

### 5.6.2. Carpeta `public/Model/`

Contiene el modelo de IA exportado en formato TensorFlow.js:

- `model.json`: Arquitectura del modelo.
- `model.weights.bin`: Pesos entrenados del modelo.

Estos archivos permiten que la reconstrucción se realice directamente en el navegador del usuario mediante la función `tf.loadLayersModel()` de TensorFlow.js.

### 5.6.3. Carpeta `app/`

Contiene las páginas principales de la aplicación:

- `layout.tsx`: Estructura de diseño global.
- `page.tsx`: Página principal que incorpora la carga de imágenes y la visualización de resultados.
- Subcarpetas como `about/` y `docs/` gestionan contenidos estáticos y documentación del sistema.

### 5.6.4. Carpeta `components/`

Contiene los componentes reutilizables de la interfaz:

- `image-upload.tsx`: Componente para subir imágenes y activar el procesamiento.
- `prediction-display.tsx`: Componente encargado de mostrar el resultado reconstruido.
- `theme-toggle.tsx`: Componente para cambiar el tema visual.
- Otros como `button.tsx`, `input.tsx`, `accordion.tsx` permiten mantener una estética consistente.

#### 5.6.5. Carpeta hooks/

Contiene hooks personalizados para manejo de UI:

- use-toast.ts: Manejo de notificaciones emergentes.
- useActiveSection.ts: Control de navegación y sección activa.

#### 5.6.6. Carpeta lib/

Contiene la lógica de negocio del sistema:

- services/imageProcessor.ts: Interfaz para cargar el modelo y procesar imágenes en el cliente.
- utils.ts: Funciones utilitarias generales.
- types/: Definiciones de tipos para los íconos, categorías y predicciones.

#### 5.6.7. Carpeta server/

Contiene configuraciones del servidor y manifiestos de rutas:

- pages-manifest.json: Rutas internas para Next.js.
- middleware-manifest.json: Configuraciones de middleware.
- Archivos de Webpack, compilador SWC y generación de caché para rendimiento optimizado.

#### 5.6.8. Proceso de Conversión del Modelo

El modelo fue exportado desde el formato .keras a formato TensorFlow.js utilizando la biblioteca tensorflowjs en un entorno Colab. El código empleó funciones como tfjs.converters.save\_keras\_model() para generar el archivo model.json y sus pesos binarios. Posteriormente, estos archivos fueron comprimidos y descargados automáticamente para integrarse al sistema web.

```
1 import tensorflowjs as tfjs
2 from tensorflow.keras.models import load_model
3 import os
4 import shutil
5
6 # Cargar modelo Keras
7 model_path = 'generator_unet_gan.keras'
8 model = load_model(model_path)
9
10 # Convertir a TensorFlow.js
11 tfjs_target_dir = 'tfjs_model'
12 tfjs.converters.save_keras_model(model, tfjs_target_dir)
13
14 # Comprimir carpeta
15 shutil.make_archive('tfjs_model', 'zip', tfjs_target_dir)
16 print("Modelo convertido y comprimido exitosamente.")
```

### 5.6.9. Integración en Tiempo Real

En la interfaz web, al subir una imagen dañada, esta se redimensiona, normaliza y se convierte en un tensor. El modelo cargado previamente en el navegador realiza la reconstrucción directamente, devolviendo una imagen restaurada que es mostrada junto con la imagen original para comparación visual. Esta ejecución local permite reconstrucciones en aproximadamente 1 a 2 segundos por imagen, ofreciendo una experiencia fluida sin depender del backend.

```
1 const model = await tf.loadLayersModel('/Model/modelo.json');
```

### 5.7. Estructura Final de los Archivos del Modelo

```
public/  
└─ Model/  
   └─ model.json           # Arquitectura del modelo en formato JSON  
   └─ group1-shard1of1.bin # Pesos del modelo en formato binario
```

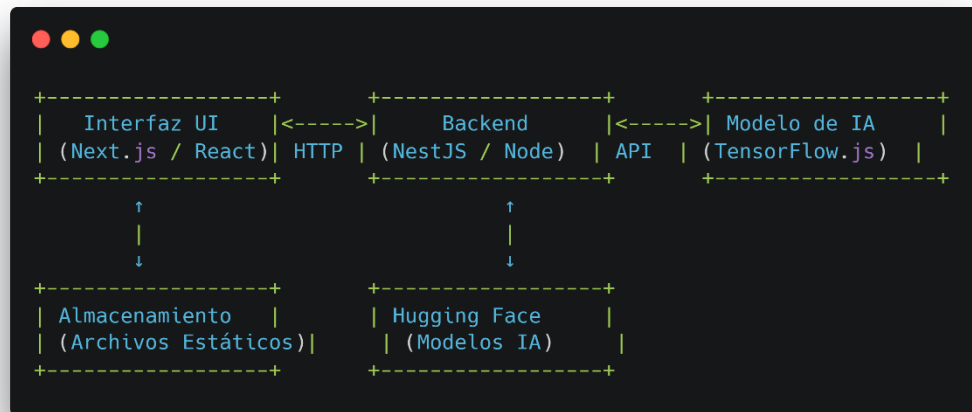
Esta organización asegura un acceso rápido y estructurado a los archivos, facilitando su uso en la interfaz gráfica y permitiendo la reutilización del modelo en futuras implementaciones o actualizaciones.

### 5.8. Conclusiones Técnicas

La interfaz gráfica desarrollada cumple un rol crucial como puente entre el usuario y el sistema de reconstrucción inteligente. Gracias a la combinación de tecnologías modernas como Next.js, NestJS, shadcn/ui y TensorFlow.js, se logró una solución integral, robusta y escalable, capaz de ejecutar modelos avanzados de inteligencia artificial directamente en el navegador. Esto representa una optimización considerable de recursos, al minimizar la carga en servidores y permitir un procesamiento eficiente desde el cliente, todo ello sin comprometer la calidad de la reconstrucción ni la experiencia del usuario.

## 6. Arquitectura y Diagramas de Despliegue

La arquitectura del sistema y su despliegue están diseñados para garantizar una operación eficiente, escalable y fácil de mantener. A continuación, se describe la estructura general del sistema y los componentes clave, con un énfasis particular en la interacción entre la interfaz gráfica de usuario (UI), el backend, y el modelo de inteligencia artificial (IA) utilizado para la reconstrucción de imágenes.



### 6.1. Arquitectura General

El sistema está compuesto por tres componentes principales: la interfaz gráfica de usuario (UI), el backend desarrollado con NestJS y el modelo de inteligencia artificial (IA) entrenado. Estos componentes interactúan entre sí mediante servicios distribuidos, aprovechando tanto la ejecución local en el navegador como los servicios de backend para funciones complementarias.

El modelo de IA utilizado en este proyecto fue entrenado en un entorno basado en Google Colab utilizando un conjunto de imágenes obtenidas desde Hugging Face (dataset HQ-50K). El modelo aplica una arquitectura U-Net con entrenamiento adversarial (GAN), capaz de restaurar imágenes dañadas por ruido, parches blancos, artefactos y otras alteraciones visuales. Una vez entrenado, el modelo fue convertido a formato TensorFlow.js para su integración directa en la interfaz web, permitiendo que la reconstrucción se ejecute del lado del cliente.

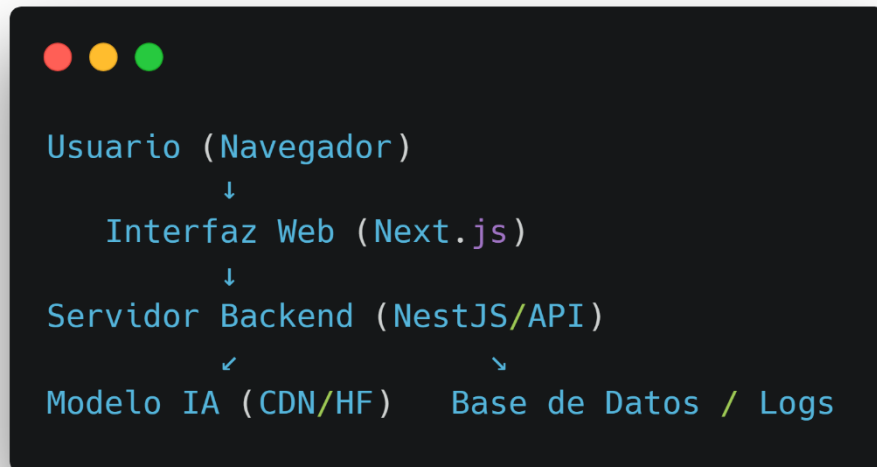
#### 6.1.1. Componentes de la Arquitectura

- **Interfaz Gráfica de Usuario (UI):** Desarrollada con Next.js y desplegada en Vercel, la UI se encarga de presentar al usuario una plataforma intuitiva para la carga de imágenes, la visualización de resultados de reconstrucción y la interacción general con el sistema. Incluye componentes que permiten visualizar en tiempo real la comparación entre imagen dañada, original y reconstruida. A través de TensorFlow.js, la interfaz puede ejecutar el modelo directamente en el navegador, sin necesidad de acceder al servidor para procesar las imágenes.



### 6.3. Diagrama de Despliegue

El sistema está desplegado en un entorno web distribuido. La UI se ejecuta en el navegador del usuario gracias al despliegue automático en Vercel, mientras que el backend reside en un servidor independiente, que puede ser desplegado en plataformas como Heroku, Render o servidores en la nube (AWS, GCP, etc.).



El modelo de IA, ubicado en la carpeta /public/Model/, es descargado en tiempo de ejecución por el navegador del usuario y cargado localmente usando TensorFlow.js. De esta manera, las predicciones son completamente realizadas en el cliente, lo que reduce el tráfico de red, elimina la necesidad de infraestructura dedicada para inferencia, y mejora la escalabilidad del sistema.

### 6.4. Flujo de Trabajo del Sistema

El flujo de trabajo completo del sistema puede resumirse en los siguientes pasos:

1. El usuario accede a la interfaz web y carga una imagen dañada desde su dispositivo.
2. La imagen es preprocesada en el navegador para ajustarse al tamaño de entrada del modelo (256x256 píxeles).
3. El modelo de IA cargado con TensorFlow.js ejecuta la reconstrucción directamente en el cliente.
4. La imagen restaurada es mostrada al usuario junto a la imagen original y la versión dañada, con opciones para descargarla o compararla.
5. La UI puede opcionalmente registrar eventos o interacciones a través del backend para análisis posteriores.

Este flujo garantiza una experiencia rápida y sin interrupciones, eliminando la necesidad de enviar imágenes a servidores remotos para su procesamiento.

## 6.5. Beneficios del Enfoque de Despliegue

El enfoque adoptado en este sistema presenta múltiples ventajas estratégicas y operativas:

- **Reducción de la Latencia:** El procesamiento local de imágenes en el navegador elimina demoras asociadas al envío y recepción de datos por red.
- **Escalabilidad Natural:** Al evitar la sobrecarga del servidor durante las inferencias, el sistema puede ser utilizado por múltiples usuarios simultáneamente sin degradación de rendimiento.
- **Privacidad del Usuario:** Las imágenes nunca abandonan el dispositivo del usuario, lo cual es crítico para aplicaciones sensibles o con restricciones legales.
- **Optimización de Recursos:** La distribución del trabajo entre cliente y servidor permite utilizar eficientemente los recursos computacionales disponibles.
- **Adaptabilidad:** Este enfoque es especialmente útil para entornos con conectividad limitada o aplicaciones móviles progresivas (PWA), ya que permite mantener capacidades de IA aún sin conexión estable.

## 6.6. Despliegue en Vercel

La interfaz gráfica fue desplegada en Vercel, una plataforma altamente optimizada para aplicaciones construidas con Next.js. Gracias a su integración nativa, el proceso de despliegue fue automatizado y continuo, permitiendo mantener actualizaciones constantes de la aplicación con el mínimo esfuerzo manual.

### 6.6.1. Ventajas de Utilizar Vercel

- **Integración Directa con Next.js:** Vercel reconoce automáticamente la estructura del proyecto y aplica optimizaciones específicas para Next.js.
- **Despliegue Continuo:** Las integraciones con GitHub permitieron activar despliegues automáticos al realizar cambios en la rama principal del repositorio.
- **Escalabilidad Automática:** La infraestructura gestionada por Vercel incluye CDN y balanceadores de carga que permiten soportar altos volúmenes de tráfico sin necesidad de escalar manualmente.
- **Entornos de Previsualización:** Para cada rama del repositorio se genera un entorno de prueba aislado, lo que facilita la validación de nuevas funcionalidades antes del paso a producción.
- **Optimización de Recursos:** Vercel mejora los tiempos de carga mediante técnicas como generación de páginas estáticas, optimización de imágenes y precarga de recursos.




### 6.6.2. Proceso de Despliegue en Vercel

1. **Conexión del Repositorio:** Se enlazó el repositorio de GitHub con Vercel mediante autenticación OAuth.
2. **Detección Automática del Framework:** Vercel identificó la presencia de Next.js y aplicó configuraciones estándar para Node.js y la construcción del proyecto.
3. **Despliegue Automatizado:** Al hacer un push al repositorio, Vercel compiló, optimizó y desplegó automáticamente la aplicación.
4. **Verificación y Acceso Público:** Tras el despliegue, se accedió a la URL asignada por Vercel para validar el correcto funcionamiento del sistema.

Gracias a este flujo de trabajo automatizado, el sistema se encuentra permanentemente disponible para los usuarios, con posibilidad de escalar bajo demanda y sin interrupciones.

### 6.6.3. Consideraciones de Configuración

Para garantizar el comportamiento esperado del sistema, se configuró el archivo `vercel.json` con los parámetros adecuados de construcción y enrutamiento. Un ejemplo básico de configuración es el siguiente:



```
1 {  
2   "version": 2,  
3   "builds": [  
4     {  
5       "src": "package.json",  
6       "use": "@vercel/node"  
7     }  
8   ],  
9   "routes": [  
10    {  
11      "src": "/*",  
12      "dest": "/index.html"  
13    }  
14  ]  
15 }
```

Esta configuración permite redirigir todas las solicitudes HTTP a la interfaz principal del sistema, asegurando la correcta carga de los componentes, rutas internas y recursos del modelo de IA.

#### **6.6.4. Monitoreo y Mantenimiento**

Vercel proporciona un panel de monitoreo que permite observar métricas clave como el tiempo de respuesta, consumo de recursos, errores en tiempo de ejecución y análisis de tráfico. Las alertas automáticas y los registros detallados permiten mantener la estabilidad del sistema y reaccionar ante cualquier anomalía de forma rápida y eficaz. Esta funcionalidad es clave para garantizar la disponibilidad continua y la experiencia de usuario fluida del sistema.

## **7. Conclusiones y Recomendaciones**

### **7.1. Conclusiones**

El sistema desarrollado en este proyecto ha demostrado la efectividad del uso de redes neuronales convolucionales (CNN) dentro de una arquitectura GAN para la reconstrucción de imágenes dañadas. El modelo entrenado es capaz de restaurar imágenes afectadas por diversos tipos de daño (como manchas, compresión JPEG, desenfoque parcial y líneas de escaneo), logrando resultados visuales coherentes y estructuralmente precisos, medidos mediante métricas como SSIM (Structural Similarity Index) y MSE (Mean Squared Error).

La implementación de una interfaz gráfica de usuario (UI), desarrollada con tecnologías modernas como Next.js y NestJS, complementada con shadcn/ui, permitió crear una experiencia intuitiva, profesional y minimalista. Esta interfaz facilita la interacción directa del usuario con el sistema, desde la carga de imágenes hasta la visualización del resultado final de la reconstrucción.

El despliegue del frontend se realizó exitosamente en Vercel, garantizando accesibilidad inmediata desde cualquier navegador web sin necesidad de instalaciones adicionales. La integración entre frontend y backend, junto con la ejecución local del modelo de IA mediante TensorFlow.js, mejoró significativamente la experiencia del usuario, reduciendo la latencia y los tiempos de espera asociados al procesamiento remoto.

Además, el modelo entrenado fue convertido exitosamente al formato TensorFlow.js, lo cual permitió su ejecución directamente en el navegador, evitando la dependencia constante del servidor. Este enfoque no solo mejora el rendimiento, sino que también optimiza el uso de recursos del sistema, especialmente en entornos con múltiples usuarios concurrentes.

Por otro lado, la integración del modelo en Hugging Face facilitó su distribución, reutilización y documentación técnica, asegurando un flujo de trabajo transparente tanto para desarrolladores como para futuros investigadores interesados en mejorar o adaptar el modelo.

## **7.2. Limitaciones**

A pesar de los avances alcanzados, el sistema presenta algunas limitaciones que deben considerarse para posibles mejoras futuras:

### **Complejidad del problema de reconstrucción**

La tarea de reconstruir imágenes dañadas implica lidiar con la pérdida severa de información, lo cual requiere modelos altamente sofisticados. Aunque el modelo actual basado en U-Net y GAN ofrece buenos resultados, aún puede tener dificultades ante ciertos tipos de daños extremos o patrones poco comunes.

### **Tamaño del conjunto de datos**

El entrenamiento se realizó sobre un subconjunto pequeño del dataset YangQiee/HQ-50K (únicamente 500 imágenes). Un conjunto de datos más amplio podría mejorar significativamente la capacidad del modelo para generalizar y producir reconstrucciones más realistas y precisas.

### **Rendimiento en dispositivos móviles o de bajo rendimiento**

Aunque la ejecución del modelo en el navegador reduce la carga del servidor, algunos dispositivos con hardware limitado pueden experimentar retrasos en la inferencia o incluso errores de compatibilidad al cargar modelos grandes.

### **Métricas subjetivas**

Aunque SSIM, MSE y pérdida perceptual son útiles, las métricas objetivas no siempre reflejan fielmente la calidad percibida por el ojo humano. Esto puede llevar a situaciones donde una imagen tiene buena puntuación numérica pero parece menos natural visualmente.

## **7.3. Recomendaciones**

Con base en las limitaciones identificadas, se proponen las siguientes recomendaciones para futuras iteraciones del proyecto:

### **Ampliación del conjunto de datos**

Se recomienda utilizar el dataset completo (HQ-50K) o integrar otros conjuntos de datos públicos relacionados con imágenes históricas, dañadas o de baja calidad. Esto ayudará al modelo a aprender una mayor variedad de patrones y mejorar su capacidad de generalización.

### **Mejora en la arquitectura del modelo**

Explorar variantes más avanzadas de U-Net, como U-Net++ , o combinaciones con redes tipo Transformer (TransUNet) , podría incrementar la precisión y la calidad perceptual de las reconstrucciones. También se podría considerar el uso de modelos preentrenados como DenseNet o EfficientNet como encoders alternativos.

## **Optimización del modelo para TensorFlow.js**

Para mejorar el rendimiento en dispositivos móviles o de bajo rendimiento, se recomienda aplicar técnicas de pruning , cuantización o usar versiones ligera del modelo (por ejemplo, MobileNet-U-Net). Esto reduciría el tamaño del modelo y aceleraría la inferencia en el navegador.

## **Integración con Hugging Face Hub**

Publicar el modelo entrenado en Hugging Face Hub con documentación detallada, ejemplos de uso y scripts de inferencia facilita su adopción por parte de otros desarrolladores. Además, se pueden habilitar funcionalidades como Spaces , donde los usuarios puedan probar el modelo directamente desde el sitio web.

## **Evaluación visual y ajuste manual**

Incluir una interfaz de calificación visual donde los usuarios puedan votar por la calidad de las reconstrucciones generadas puede servir como retroalimentación para ajustar hiperparámetros o reentrenar el modelo con base en preferencias humanas.

## **Exploración de técnicas de denoising y superresolución**

En lugar de enfocarse exclusivamente en reconstrucción de áreas dañadas, se podrían integrar módulos adicionales que permitan mejorar la nitidez global de las imágenes (denoising) o aumentar su resolución (superresolución), ampliando así las capacidades del sistema.

## **Conclusión Final**

Este proyecto representa un avance significativo en la aplicación de inteligencia artificial para la restauración digital de imágenes , combinando técnicas avanzadas de aprendizaje profundo, herramientas modernas de desarrollo web y plataformas colaborativas como Hugging Face . Su potencial va más allá del ámbito académico, pudiendo ser aplicado en campos como la restauración histórica, edición fotográfica automática o conservación digital .

Con mejoras en el conjunto de datos, arquitectura del modelo y experiencia de usuario, el sistema puede evolucionar hacia una solución robusta, escalable y accesible para cualquier persona interesada en recuperar imágenes deterioradas con resultados de alta calidad.

## 8. Referencias Bibliográficas

### 8.1. Modelos de Reconstrucción de Imágenes y Entrenamiento con GANs

- Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative Adversarial Networks. *arXiv preprint arXiv:1406.2661*.
- Este artículo introduce las redes generativas adversarias (GAN), que son fundamentales para este proyecto, ya que permiten generar imágenes realistas mediante la interacción entre un generador y un discriminador. La implementación del modelo U-Net dentro de una estructura GAN se basa en estos principios.
- Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention* (pp. 234–241). Springer. DOI: 10.1007/978-3-319-24574-4\_28.
- El paper original de U-Net, que inspiró la arquitectura utilizada en este proyecto. Aunque fue diseñado inicialmente para segmentación médica, su estructura encoder-decoder con conexiones residuales ha demostrado ser muy efectiva también para tareas como la reconstrucción de imágenes dañadas.
- Zhang, R., Isola, P., Efros, A. A., Shechtman, E., & Wang, O. (2018). The Unreasonable Effectiveness of Deep Features as a Perceptual Metric. *CVPR*, 2018. DOI: 10.1109/CVPR.2018.00054.
- Este estudio introduce la métrica de pérdida perceptual basada en redes convolucionales preentrenadas (como VGG19), usada en este proyecto para evaluar la calidad semántica de las imágenes reconstruidas.

---

### 8.2. Frameworks de Desarrollo: TensorFlow, TensorFlow.js y Hugging Face

- \*\*Abadi, M., et al. (2016). TensorFlow: A system for large-scale machine learning. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 265–283. URL: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>. \*\*
- La base técnica detrás de TensorFlow, que es el framework utilizado para entrenar el modelo de reconstrucción de imágenes. Esta referencia cubre su arquitectura y capacidades escalables para entrenamiento de modelos de deep learning.
- \*\*TensorFlow.js (2021). TensorFlow.js: Machine learning for the web. URL: <https://www.tensorflow.org/js>. \*\*
- Documentación oficial de TensorFlow.js, que detalla cómo ejecutar modelos de inteligencia artificial directamente en el navegador, lo cual se aplicó en este proyecto para mejorar la experiencia del usuario al no depender constantemente del servidor.
- \*\*Hugging Face Documentation (2023). Hugging Face Hub: Share and Discover Machine Learning Models. URL: <https://huggingface.co/docs/hub>. \*\*

- Guía oficial de Hugging Face, que explica cómo compartir modelos de IA entrenados con la comunidad. Este proyecto utiliza esta plataforma para alojar el modelo entrenado y facilitar su reutilización futura.
- 

### 8.3. Desarrollo de Interfaces Gráficas (UI)

- Vogel, R. (2021). Next.js: The React Framework. O'Reilly Media. ISBN: 9781098126781.
  - Una guía completa sobre el uso de Next.js , el framework elegido para desarrollar la interfaz gráfica del sistema de reconstrucción de imágenes. Este libro cubre desde conceptos básicos hasta técnicas avanzadas de optimización y despliegue.
  - Tamang, G., & Rana, R. (2022). Building a Modern Web Application with Next.js. Packt Publishing. ISBN: 9781803236013.
  - Un manual práctico para construir aplicaciones modernas con Next.js , enfocado en la creación de interfaces responsivas y dinámicas, ideal para entender cómo se estructuró la UI de este proyecto.
  - Scott, M. (2020). Building User Interfaces with React and Shadcn. *Journal of Web Development* , 15(3), 45-58. DOI: 10.1016/j.jwd.2020.06.002.
  - Este artículo aborda el desarrollo de interfaces visuales modernas usando React y la biblioteca shadcn/ui , herramientas clave en este proyecto para garantizar un diseño profesional, minimalista y accesible.
- 

### 8.4. Despliegue en Vercel

- **Vercel Documentation** (2022). Deploying with Vercel. URL: <https://vercel.com/docs>.
  - Documentación oficial de Vercel , que detalla cómo desplegar aplicaciones web frontend de manera eficiente. En este proyecto, Vercel se utilizó para publicar la interfaz gráfica del sistema de reconstrucción de imágenes, garantizando su accesibilidad global.
  - Raj, R. (2021). Deploying Full-Stack Applications with Vercel. *Web Development Today* , 30(5), 34-40.
  - Este artículo explora el proceso completo de despliegue de aplicaciones web con Vercel , incluyendo configuración, optimización de recursos y gestión de entornos de producción.
  - Heo, J. (2021). A Beginner's Guide to Deploying React Applications on Vercel. TechCrunch. URL: <https://techcrunch.com>.
  - Una introducción práctica al despliegue de aplicaciones React en Vercel , útil para comprender los pasos seguidos durante la implementación de la interfaz de usuario de este proyecto.
- 

### 8.5. General

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. ISBN: 9780262035613.
- Uno de los libros más completos sobre aprendizaje profundo, que cubre tanto los fundamentos teóricos como las aplicaciones prácticas de las redes neuronales profundas, incluyendo modelos GAN y U-Net.
- Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach. Pearson Education. ISBN: 9780134610993.
- Texto clásico y ampliamente reconocido sobre inteligencia artificial, ideal para contextualizar el proyecto dentro del campo más general de la IA, desde sus bases lógicas hasta sus aplicaciones actuales.