



Escuela Técnica Superior de
Ingeniería Informática

**GRADO EN INGENIERÍA INFORMÁTICA INGENIERÍA
DEL SOFTWARE**

Re-escalado y análisis de imágenes digitales

Realizado por

**Barea Jiménez, Antonio
Cabezas Villalba, Juan Pablo
Camero Borrego, Tomás**

Profesores
Medrano Garfia, Belén

Asignatura
Procesamiento de imágenes digitales

Departamento
Lenguajes y Sistemas Informáticos

Agradecimientos

Nos gustaría comenzar expresando nuestro más profundo agradecimiento a nuestros profesores y universidad por permitirnos explorar y desarrollar nuestro conocimiento en el campo del procesamiento digital de imágenes.

Inicialmente agradecer a DotCsv por sus videos de divulgación de la materia [9], los artículos de documentación de Tensorflow sobre pix2pix y a Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros por la investigación de redes convolucionales que ha permitido la facilidad en la fabricación de la arquitectura [7].

A lo largo del proyecto, los tres hemos trabajado en estrecha colaboración y nos gustaría reconocer nuestras contribuciones individuales y compartidas.

Antonio, gracias por tu habilidad y experiencia en el campo del procesamiento digital, tus valiosos aportes en el diseño y análisis de algoritmos han enriquecido significativamente nuestro trabajo.

Pablo, te agradecemos por ser un excelente líder de equipo y mantenernos enfocados y motivados durante el proceso. Tu dedicación y pasión por la tecnología se han visto reflejadas en la calidad de nuestro trabajo conjunto.

Y Tomas, como tercer integrante del equipo, trabajando en estrecha colaboración con Antonio y Pablo, proporcionando el conocimiento y experiencia para contribuir al éxito de este proyecto.

También nos gustaría agradecer a nuestras familias y amigos por su apoyo incondicional y comprensión durante el desarrollo de este trabajo. Su amor y ánimo nos han dado la energía necesaria para enfrentar los desafíos y superar los obstáculos a lo largo del proceso.

En conjunto, hemos aprendido y crecido gracias a esta experiencia, y estamos orgullosos de lo que hemos logrado este proyecto.

Resumen

Este trabajo demuestra la aplicación exitosa de una red GAN, especialmente la arquitectura Pix2Pix, para generar y mejorar imágenes de gatos a partir de conjuntos de datos de imágenes de gatos, analizando como actúa dicha red cuando se le aplican diferentes tipos de pre-procesamiento para empeorar las imágenes de entrenamiento.

Esto destaca la capacidad de GAN para aprender a crear imágenes realistas y mejorar la calidad de imagen en un área específica.

Palabras clave: GAN, Calidad, Pix2Pix, Preprocesamiento

Abstract

This work demonstrates the successful application of GAN network, especially the Pix2Pix architecture, to generate and improve cat images from cat image data sets, analyzing how this network acts when different types of pre-processing are applied to it to worsen the images. training images.

This highlights GAN's ability to learn how to create realistic images and improve image quality in a specific area.

Keywords: GAN, Quality, Pix2Pix, Preprocessing

Índice general

| | |
|---|-----------|
| 1. Introducción | 1 |
| 2. Planteamiento Teórico | 2 |
| 3. Implementación | 3 |
| 3.1. Dependencias | 3 |
| 3.2. Preprocesado de datos | 3 |
| 3.2.1. Creación de imágenes de entrenamiento | 4 |
| 3.2.2. Elección de número de imágenes | 4 |
| 3.2.3. Aumento de datos para imágenes de entrada y salida | 5 |
| 3.3. Implementación de la GAN | 6 |
| 3.3.1. Generador | 6 |
| 3.3.2. Discriminador | 7 |
| 3.3.3. Funciones de pérdida y optimizadores para GAN y generación de imágenes | 8 |
| 3.4. Entrenamiento | 8 |
| 4. Experimentación | 10 |
| 4.1. Preparación del entorno de experimentación | 10 |
| 4.2. Ejemplos comentados | 11 |
| 4.2.1. Gauss | 11 |
| 4.2.2. Sal y pimienta | 14 |
| 4.2.3. Transformación logarítmica | 15 |
| 4.2.4. Sobel | 19 |
| 4.3. Pruebas realizadas | 23 |
| 5. Conclusiones | 24 |
| 6. Tabla de tiempos | 26 |
| 7. Bibliografía | 27 |

Índice de figuras

| | | |
|-------|--|----|
| 4.1. | Ejemplo de imagen de entrenamiento junto a imagen real | 11 |
| 4.2. | Ejemplo de resultado real | 12 |
| 4.3. | Ejemplo de resultados varios | 12 |
| 4.4. | Ejemplo de resultados con un input más borroso | 13 |
| 4.5. | Ejemplo de imagen de entrenamiento junto a imagen real | 14 |
| 4.6. | Ejemplo de imagen de entrenamiento junto a imagen real | 14 |
| 4.7. | Visualización con Transformación de logaritmo | 15 |
| 4.8. | Resultados I | 16 |
| 4.9. | Resultados II | 17 |
| 4.10. | Resultados III | 18 |
| 4.11. | Visualización con Sobel | 19 |
| 4.12. | Visualización con Sobel con rotación | 19 |
| 4.13. | Resultados I | 20 |
| 4.14. | Resultados II | 21 |
| 4.15. | Resultados III | 22 |

1. Introducción

El rápido desarrollo de las tecnologías de procesamiento digital y el aumento en la cantidad de imágenes disponibles en Internet han creado la necesidad de desarrollar métodos eficientes y efectivos para mejorar la calidad y su posterior análisis. En este contexto, las redes antagónicas generales (GAN) [3] han demostrado ser una herramienta prometedora para resolver estos problemas, capaces de generar y transformar imágenes con resultados impresionantes. En particular, la arquitectura Pix2Pix [10] ha demostrado ser particularmente útil para crear y mejorar imágenes en una variedad de campos. En este trabajo, nos centraremos en escalar y analizar imágenes de diversos gatos (en diferentes contextos) utilizando GAN y la arquitectura Pix2Pix. Nuestro objetivo es explorar el potencial de estas técnicas para generar imágenes de alta calidad e investigar cómo responden a diferentes condiciones de preprocesamiento que degradan las imágenes de entrenamiento. Comenzaremos con una descripción general de los conceptos básicos de las arquitecturas GAN y Pix2Pix y cómo se relacionan con las imágenes digitales.

A continuación, describimos el procedimiento de recopilación de datos y los métodos de pretratamiento utilizados en nuestro estudio. Luego presentaremos nuestro método de implementación y entrenamiento de GAN, y discutiremos los resultados obtenidos usando diferentes tipos de preprocesamiento de imágenes de entrenamiento.

Finalmente, discutimos los resultados y conclusiones de nuestro estudio, destacando las capacidades y limitaciones de las arquitecturas GAN y Pix2Pix en el escalado y análisis de imágenes. Además, identificaremos oportunidades para futuras investigaciones y posibles aplicaciones en áreas relacionadas con la imagen digital.

2. Planteamiento Teórico

En este artículo, se presenta una aplicación de redes neuronales adversas generativas (GAN) para crear y mejorar imágenes de gatos a partir de un conjunto de datos de imágenes de gatos.

La implementación utiliza la arquitectura Pix2Pix [10], que es una GAN basada en el modelo decodificador-decodificador con conectividad de salto.

Para el proceso de preparación de datos de disminuir la calidad de las imágenes, se usará;

- **Filtro de Gauss**
- **Transformación logarítmica**
- **Ruido de sal y pimienta**
- **Filtro de Sobel**

Cada tipo de preprocessado se hará de manera independiente, entrenando la red para cada uno, con el fin de ver las similitudes y diferencias entre los distintos entrenamientos. Estas transformaciones se han visto de manera detallada en [1].

Como parte del procesado común, se realiza un conjunto de transformaciones de imágenes, como rotación aleatoria, zoom y volteo, para aumentar la diversidad de datos y mejorar la solidez del modelo.

Luego las partes mas importantes de la red son las siguientes:

- **Generador:** El generador de arquitectura Pix2Pix consta de una serie de capas de convolución sobre-muestreadas, y con conexiones entre ellas.
- **Discriminador:** El discriminador es una red convolucional que toma tanto la entrada como la imagen generada y toma una decisión sobre si la imagen generada es realista o no. La función de pérdida de entropía cruzada binaria se utiliza para calcular la pérdida del discriminador y la pérdida del generador. Además, se incluye una pérdida L1 para garantizar que las imágenes generadas sean similares a las imágenes de destino.

Para la optimización del modelo se utiliza el optimizador Adamax para el generador y el discriminador. El entrenamiento del modelo se realiza durante varias épocas, actualizando los pesos de la red neuronal en función de las funciones de pérdida calculadas. Durante el entrenamiento, se generan imágenes de muestra para visualizar el progreso del modelo. Al final del entrenamiento, el generador puede generar imágenes de gatos mejoradas a partir de las imágenes de entrada.

3. Implementación

Nuestro código se divide en 4 partes, *Dependencias*, *Procesado de datos*, *Implementación de la GAN* y *Entrenamiento*. A continuación se detallará cada una.

3.1. Dependencias

En primer lugar, se importan las dependencias necesarias para ejecutar el proyecto. Adicionalmente existe una celda por si se quiere comprobar que se tenga CUDA [8] instalado con el fin de aligerar el entrenamiento del modelo (recomendamos esa opción, aunque la instalación es algo tediosa). Los pasos generales consisten en la instalación de los drivers de nvidia, además de las librerías de CUDA toolkit + cuDNN y TensorFlow 2.6 [2] mínimo con python 3.7 [5] junto a las librerías de compilación de Visual Studio para C y creando un entorno virtual dentro de anaconda e instalando:

```
pip install tensorflow-gpu==2.6.0
```

Debería ser suficiente. Existen muchas variantes y en nuestro caso no ha funcionado hasta que se ha ejecutado el proyecto dentro de la carpeta del entorno virtual. Se recomienda seguir algún tutorial.

Este entorno se ha usado en la renderización de los resultados de Gauss y de sal y pimienta.

Por otra parte, algunos miembros también han realizado la experimentación usando Google Colab [6] la mayoría de veces, debido a que implementa uso de GPU como una de sus opciones y es gratis y accesible desde cualquier lado. Es por ello que, por ejemplo el notebook de Sobel y transformación logarítmica tienen algunas celdas adicionales para el uso en Google Colab. Su configuración es relativamente sencilla, se tendrán que ejecutar las primeras celdas de dichos notebooks enfocados en Google Colab, siendo una de ellas la clonación del proyecto raíz desde GitHub (copiándose la carpeta del proyecto en la raíz del sistema de archivos del Colab) y otra para la conexión con Google Drive (donde se guardarán las fotos que el modelo vaya creando). Una vez hecho esto, habrá que mover la carpeta *org_cats* a la raíz del Colab, y crear una nueva con el nombre de *trn_cats*. Por último, habrá que crear una carpeta en la raíz de la cuenta que hayamos conectado Google Drive, con el nombre de output (si es el notebook de T. Logarítmica) o output2 (si es el notebook de Sobel).

3.2. Preprocesado de datos

En esta sección se aplican las diferentes técnicas de preprocesado a un dataset origen indefinido (en nuestro caso, el uso de una variedad de imágenes de gatos [4] para crear el set de entrenamiento y pruebas correspondiente. Podemos dividir esta sección en:

3.2.1. Creación de imágenes de entrenamiento

Con el dataset ya en una carpeta accesible, la vamos recorriendo, leyendo cada imagen con opencv, para luego poder aplicarle uno de los tipos de preprocesado que hemos comentado antes. La manera de actuar de cada uno es casi similar:

- **Filtro de Gauss:** La aplicación del filtro de Gauss resulta en una tarea muy sencilla, una vez leída la imagen mediante la aplicación de la función `opencv.GaussianBlur(img, (0, 0), sigmaX=sigma, sigmaY=sigma)` ajustando la variable *sigma* al grado de distorsión buscado. El *sigma* usado ha sido inicialmente de 7 y se redujo a 5.
- **Transformación logarítmica:** Para aplicar la transformación logarítmica, primero debemos leer la imagen a color. Una vez leída la imagen, separamos los 3 canales de color los cuales nombraremos como b, g, r (haciendo referencia, respectivamente, a los colores blue, green, red) Posteriormente, mediante la función que nos ofrece numpy llamada `random.uniform`, alpha tomara un valor entre 0.01 y 10.0. Finalmente, a cada canal de color de la imagen le aplicamos la formula de la transformada logarítmica la cual es $(255.0/\log(1+255)) * \log(1+\alpha+\text{canal})$ y unimos dichos canales mediante el comando de opencv llamado `merge`.
- **Ruido de sal y pimienta:** Para aplicar este tipo de ruido, primero se calculan los píxeles totales de la imagen y se multiplican por un porcentaje para averiguar primeramente el grado de ruido. El procedimiento siguiente es simple, se escoge un pixel aleatorio y de manera aleatoria o según un porcentaje elegido, se insertan en la imagen pixeles blancos y negros dejando una nube aleatoria haciendo interferencia.
- **Filtro de Sobel:** Una vez tengamos la imágenes en una variable (leída), la transformamos a una imagen en escala de grises (necesario para aplicar Sobel para detectar bordes). Luego una vez elegido el eje en el que vamos a aplicar el operador, ya sea en el eje X, Y o ambos, de manera aleatoria para cada imagen, usando una función de la librería `random` con la función `choice`. Luego solo queda aplicar el filtro usando la función de opencv `opencv.Sobel` en el eje que haya tocado.

Luego de aplicar cada preprocesado, las imágenes se guardan en una carpeta previamente creada en el directorio de trabajo.

3.2.2. Elección de número de imágenes

El número de imágenes totales a repartir entre entrenamiento y testeo ha sido de 1000 debido a la cantidad que supone y el coste computacional que supone añadir mas. Primero con la función `round` calculamos un 80 % del total para usarlo para el entrenamiento. Después, con la función `shuffle`, reordenamos una variable con todas las imágenes, para garantizar que el conjunto de datos esté bien mezclado. Por último, se eligen la que serán las imágenes de entrenamiento, que serán desde la primera imagen del conjunto mezclado, hasta el 80 % calculado antes (en nuestro caso, 800 imágenes). El resto serán las de pruebas.

3.2.3. Aumento de datos para imágenes de entrada y salida

Este bloque de código define varias funciones de preprocessamiento de datos e incrementales para las imágenes de entrada y salida de un modelo de aprendizaje profundo. El propósito de estas funciones es preparar imágenes para usarlas en el entrenamiento y la evaluación de modelos, y mejorar la confiabilidad y el rendimiento del modelo exponiéndolo a cambios de entrada durante el entrenamiento.

3.2.3.1. Reescalado

Las imágenes de entrada y salida se redimensionan a un tamaño específico mediante la función de redimensionamiento. Esta característica es útil para garantizar que todas las imágenes tengan el mismo tamaño antes de transferirlas al modelo. El ancho y la altura se establecen en 512 x 512 porque ese es el tamaño que nos dio los mejores resultados después de la prueba.

3.2.3.2. Normalización

La función de normalización se utiliza para normalizar las imágenes de entrada y salida. La imagen se escala de -1 a 1 dividiendo cada píxel por 127,5 y luego restando 1. La normalización es importante para garantizar que las entradas del modelo coincidan, lo que simplifica el proceso de entrenamiento.

3.2.3.3. Jitter aleatorio

Durante el entrenamiento, se aplica una técnica de *jitter* aleatorio a las imágenes de entrada y salida utilizando la función ***random_jitter***. Esta técnica incluye:

- Reescalado temporal a un tamaño ligeramente más grande.
- Apilamiento de las imágenes de entrada y salida en un solo tensor.
- Recorte aleatorio de las imágenes apiladas al tamaño original.
- Volteo horizontal aleatorio de las imágenes.

Estas transformaciones introducen variaciones en los datos de entrenamiento y ayudan a mejorar la generalización del modelo al enfrentarse a datos no vistos. Para Sobel cabe destacar que hay una linea de código adicional debido al número de canales que tiene al ser en escala de grises. Para que funcione el código se tuvo que añadir 2 canales mas a la imagen para que fuese RGB.

3.2.3.4. Carga de imágenes

Las funciones `load_train_image` y `load_test_image` cargan las imágenes de entrenamiento y prueba, respectivamente. Estas funciones aplican solo las transformaciones

apropiadas, incluido el jitter aleatorio, a las imágenes de entrenamiento. Las imágenes de prueba no son aleatorias.

3.2.3.5. Visualización

Finalmente, se muestra un ejemplo de una imagen de entrenamiento y su resultado esperado después de aplicar las transformaciones. Esto ayuda a verificar que las transformaciones se hayan aplicado correctamente y proporciona una vista previa de al resultado que se quiere llegar una vez se utilice en el modelo de aprendizaje profundo.

3.2.3.6. Creación de conjuntos con TensorFlow

Finalmente, los conjuntos de datos de entrenamiento y prueba se generan a partir de la lista de imágenes. Las funciones de carga de imágenes predefinidas `load_train_image` y `load_test_image` se utilizan para aplicar preprocessamiento y mejora de datos a las imágenes de entrada y salida. Los conjuntos de datos resultantes están destinados a usarse en la capacitación y evaluación de un modelo de aprendizaje profundo.

3.3. Implementación de la GAN

3.3.1. Generador

El generador se encarga de generar imágenes a partir del espacio latente o ruido aleatorio. La estructura de este generador se basa en la arquitectura U-Net, que es una red neuronal convolucional (CNN) especializada en tareas de segmentación de imágenes y generación condicional de imágenes. La arquitectura U-Net consta de una serie de capas de muestreo descendente seguidas de capas de muestreo ascendente, formando una estructura en forma de U.

1. **Definición de las funciones de muestreo descendente y ascendente:** Se definen dos funciones, `downsampling` y `upsampling`, para generar bloques de capas para `downsampling` y `upsampling`, respectivamente. Estos bloques de capas incluyen capas de convolución y decodificación con diferente número de filtros y el uso de adición de normalización y eliminación por lotes. Estas funciones facilitan la arquitectura del generador de edificios al facilitar la adición y modificación de bloques de capas.
2. **Creación de la arquitectura del generador:** La función `Generador` crea una arquitectura de generador completa utilizando funciones predefinidas de submuestreo. La arquitectura consta de una pila de muestreo descendente (`down_stack`) y una cascada de muestreo ascendente (`up_stack`).
3. **Conexión de las capas del generador:** Las clases de submuestreo y sobre-muestreo están conectadas por un bucle `for` que recorre ambas pilas de clases y las combina mediante la función `Concatenar`. Esta característica permite el uso

de características aprendidas en capas reducidas en las capas correspondientes reducidas, mejorando la capacidad del generador para generar imágenes de alta calidad.

4. **Capa final de salida:** Se agrega una capa de decodificación final con tres filtros y una función de activación de tangente hiperbólica (\tanh) al final de la arquitectura del generador. Esta capa convierte las funciones aprendidas en una imagen de salida de tres canales (RGB).
5. **Creación del modelo de generador:** El modelo de construcción se genera utilizando la clase de modelo TensorFlow con las entradas y salidas definidas anteriormente. Esta arquitectura de generador ahora se puede usar para generar imágenes espaciales latentes o ruido aleatorio en contextos GAN.

3.3.2. Discriminador

El discriminador se encarga de distinguir entre la imagen real y la imagen generada. La estructura de este discriminador es una red neuronal convolucional (CNN) que toma como entrada dos imágenes: la imagen de entrada y la imagen generada (generada por el generador).

1. **Definición de las entradas del discriminador:** El discriminador tiene dos entradas: `ini` (imagen de entrada) y `gen` (imagen generada). Ambas entradas son de tamaño indeterminado, lo que permite procesar imágenes de diferentes tamaños.
2. **Concatenación de las entradas:** Las dos imágenes de entrada están conectadas a lo largo del eje de los canales con la función concatenada. Esto permite la discriminación contra ambas imágenes como una sola entrada.
3. **Creación de la arquitectura del discriminador:** La arquitectura del discriminador se construye utilizando una función de reducción de muestreo predefinida. Se aplica una serie de capas de reducción de resolución con un número variable de filtros y normalización por lotes a la entrada combinada. Esta estructura simplifica la arquitectura del discriminador y facilita agregar y modificar bloques de clase.
4. **Capa final de salida:** A la arquitectura discriminante se le agrega la capa de convolución final con filtro y kernel de tamaño 4. Esta capa tiene como objetivo convertir las características aprendidas en una única salida que representa la probabilidad de que la imagen generada sea verdadera o falsa.
5. **Creación del modelo de discriminador:** El modelo del discriminador se genera utilizando la clase Model de TensorFlow con las entradas y salidas definidas anteriormente. Esta arquitectura del discriminador se puede usar para distinguir entre imágenes reales e imágenes generadas en el contexto GAN.

3.3.3. Funciones de pérdida y optimizadores para GAN y generación de imágenes

Este bloque de código define las funciones de pérdida para el generador y el discriminador, así como los optimizadores utilizados para entrenar ambos componentes. Además, está disponible la función para crear y mostrar imágenes a partir del modelo entrenado.

Primero, se crea un objeto con pérdida utilizando la función Keras BinaryCrossentropy. Esta función de pérdida se utilizará para calcular las pérdidas del discriminador y del oscilador. Luego se define la función de pérdida del discriminador, que toma dos argumentos: la salida del discriminador para la imagen real y la imagen generada. La función calcula la pérdida real y la pérdida generada usando entropía cruzada binaria, luego suma ambas pérdidas para obtener la pérdida total de la variable discriminante.

A continuación, se define una función de pérdida del generador, que toma tres argumentos: la salida del discriminador para la imagen generada, la imagen generada y la imagen objetivo (real). La función calcula la pérdida GAN usando entropía cruzada binaria y también calcula la pérdida L1 entre la imagen generada y la imagen de destino. La pérdida total del generador se calcula como la suma de las pérdidas GAN y L1 multiplicada por el factor de ponderación (LAMBDA).

El optimizador del generador y el discriminador se generaron luego utilizando el optimizador Keras Adam con un factor de aprendizaje de 2e-4 y un factor de amortiguamiento beta_1 de 0,5. Luego se crea un objeto de punto de control para almacenar y cargar el estado del optimizador, generador y discriminador durante el entrenamiento. Finalmente, se define la función generate_image, acepte el modelo, ingrese la prueba, las imágenes objetivas y, no necesariamente el nombre del archivo para la protección y el indicador de visualización de imágenes. La función utiliza el modelo para crear pronósticos a partir de pruebas y visualización de registros, imágenes objetivas e imágenes creadas en gráficos. Si se especifica un nombre de archivo, la imagen generada se guardará en el disco.

3.4. Entrenamiento

Para terminar, se define una función llamada ***rain_step*** que realiza un paso de entrenamiento de GAN y una función de entrenamiento que realiza todo el proceso de entrenamiento durante un número específico de épocas, en nuestro caso 100 épocas es el número que hemos elegido.

La función train_step toma una imagen de entrada y una imagen de destino (real) como argumentos. Utiliza GradientTape de TensorFlow para calcular automáticamente los gradientes de pérdida del generador y del discriminador. Primero, la imagen de salida se genera usando el generador en modo de entrenamiento. A continuación, se obtiene la salida del discriminador tanto para la imagen generada como para la imagen de destino.

A continuación, se calculan las pérdidas del discriminador y del generador utilizando las funciones de pérdida definidas anteriormente. Luego, utilizando bandas de gradiente, las pendientes de pérdida del generador y del discriminador se calculan frente a sus variables entrenables. Finalmente, el gradiente se aplica utilizando un optimizador predefinido para actualizar el generador y los pesos discriminantes. La función de entrenamiento toma un conjunto de datos y el número de épocas como argumentos. Para cada época, se itera sobre el conjunto de datos y llama a ***train_step*** con cada imagen de entrada y su imagen de destino correspondiente.

Después de cada época, las imágenes se generan usando un generador entrenado para las primeras 20 imágenes en el conjunto de datos de prueba y renderizadas usando la función ***generate_images*** definida anteriormente.

Finalmente, se llama a la función de entrenamiento con el conjunto de datos de entrenamiento y el número de épocas para entrenar el modelo.

4. Experimentación

En esta sección, nuestro objetivo es analizar y discutir el rendimiento del modelo Pix2Pix en diferentes escenarios y pruebas utilizando las diferentes técnicas de preprocesamiento. Para realizar esta estimación, entrenaremos un modelo con los mismos hiperparámetros, pero aplicaremos diferentes métodos de preprocesamiento de imágenes a los datos de entrada. Estos métodos incluyen filtro Sobel, ruido de sal y pimienta, transformación logarítmica y filtrado gaussiano. Al examinar el rendimiento del modelo en estos diferentes escenarios, podemos comprender mejor cómo estos métodos afectan los resultados finales y el rendimiento general del modelo.

4.1. Preparación del entorno de experimentación

Para llevar a cabo la experimentación, hemos configurado el entorno con los siguientes hiperparámetros y ajustes:

- **Optimizador:** Adam
- **Tasa de aprendizaje del optimizador Adam para el generador y el discriminador:** 2e-4
- **Número de imágenes a elegir:** 1000
- **Porcentaje de imágenes de entrenamiento:** 80 %
- **Beta_1 para el optimizador Adam:** 0.5
- **Función de pérdida:** BinaryCrossentropy con logits
- **Lambda (peso para la pérdida L1 en la función de pérdida del generador):** 100
- **Número de épocas de entrenamiento:** 26
- **Tamaño del lote:** 1
- **Inicializador de pesos para las capas convolucionales:** Distribución normal con media 0 y desviación estándar de 0.02
- **Capas de Batch Normalization en el generador y el discriminador**
- **Función de activación LeakyReLU en las capas de submuestreo con alfa 0.2**
- **Función de activación ReLU en las capas de sobremuestreo**
- **Función de activación tangente hiperbólica en la última capa del generador**
- **Filtros en las capas de submuestreo (downsample) y sobremuestreo (upsample) en el generador:**

- Capas de submuestreo: 64, 128, 256, 512, 512, 512, 512, 512
- Capas de sobremuestreo: 512 (con dropout), 512 (con dropout), 512 (con dropout), 512, 256, 128, 64

4.2. Ejemplos comentados

4.2.1. Gauss

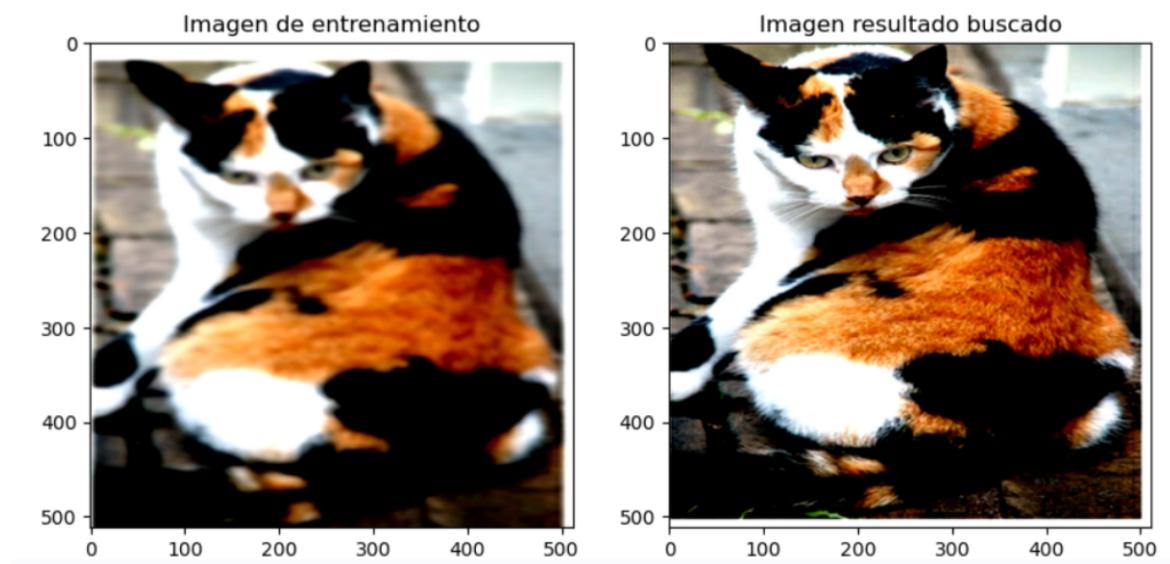


Figura 4.1: Ejemplo de imagen de entrenamiento junto a imagen real

Observamos como en [4.1](#), de forma sutil el filtro de Gauss emborrorna las siluetas y reduce el contraste de la imagen original empeorando su calidad de gran manera y siendo que en determinados casos se pierde el objetivo. Esta prueba se concentra principalmente en comprobar si la red es capaz de trabajar a pequeña escala y no sólo en la corrección de grandes características.



Figura 4.2: Ejemplo de resultado real

Como se puede distinguir, el resultado es prácticamente idéntico a la imagen original antes de hacer el filtrado por lo que podemos considerar que el resultado es muy acertado y se ha mejorado la calidad de la imagen

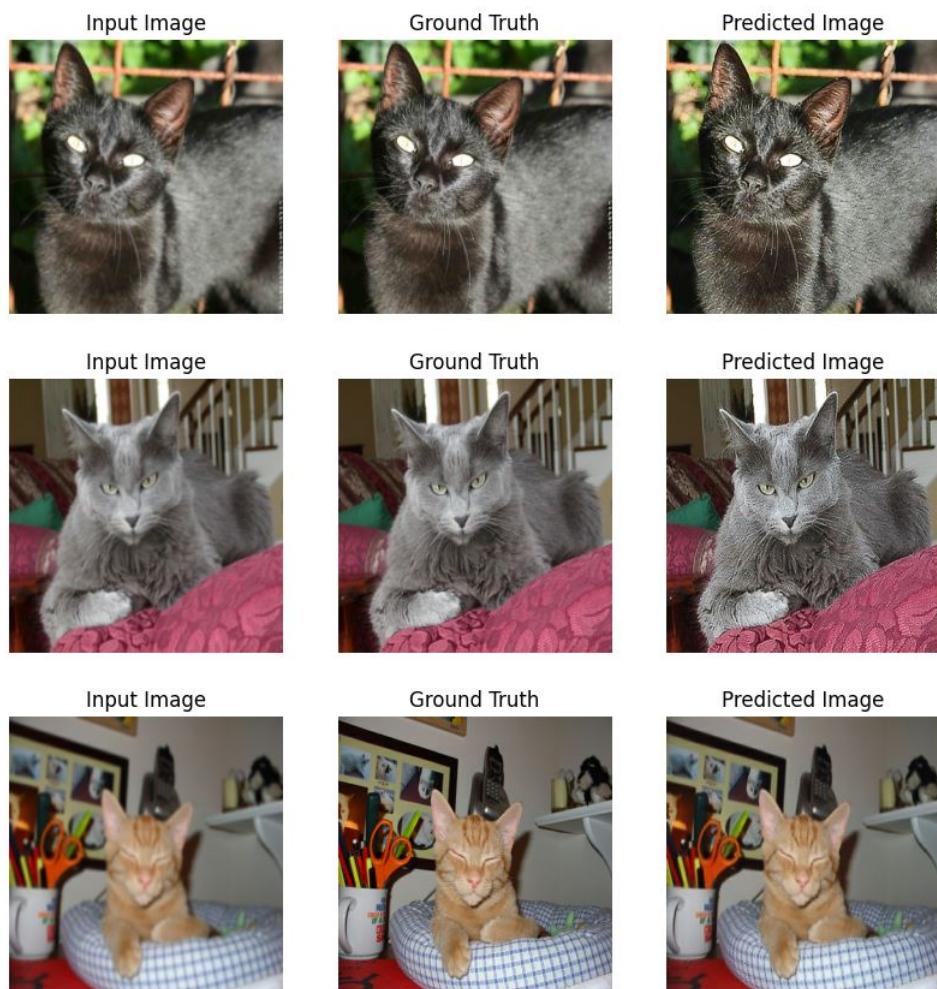


Figura 4.3: Ejemplo de resultados varios

A pesar de que la imagen input está convenientemente poco emborronada, la mejora es notable siendo que en ocasiones, como en la segunda fila de la imagen 4.4 en la que al elevar más el contraste queda como resultado una foto quizás más visual que la original.

Inicialmente, se intentó el mismo resultado con un sigma más elevado, de 7 consiguiendo resultado lógicamente peores:



Figura 4.4: Ejemplo de resultados con un input más borroso

Las causas pudieron ser:

1. **Pérdida de información de alta frecuencia:** El filtro Gaussiano se utiliza comúnmente para suavizar una imagen y eliminar detalles de alta frecuencia. Esto implica que cierta información fina puede perderse en el proceso, lo que dificulta la reconstrucción precisa de la imagen original.
2. **Reducción de contraste:** Al aplicar un filtro Gaussiano, se produce una difuminación en la imagen, lo que puede resultar en una reducción del contraste entre diferentes elementos. Esto puede dificultar la distinción de detalles importantes en la imagen original al intentar reconstruirla.
3. **Artefactos de borde:** El filtro Gaussiano tiende a producir efectos borrosos alrededor de los bordes de los objetos en una imagen. Estos artefactos pueden dificultar la reconstrucción precisa de los bordes y contornos originales.

4.2.2. Sal y pimienta

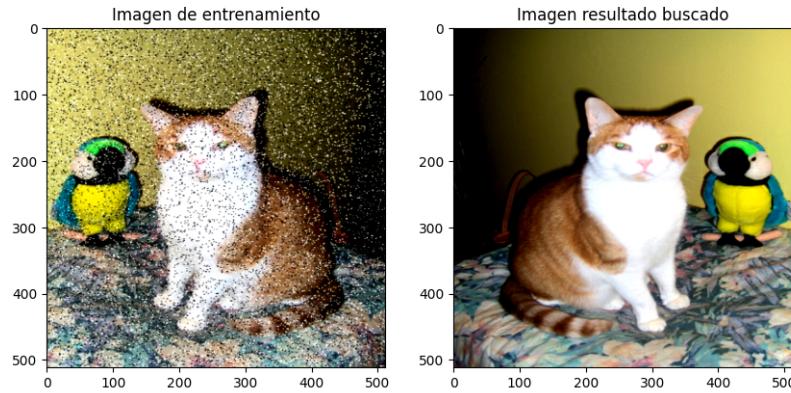


Figura 4.5: Ejemplo de imagen de entrenamiento junto a imagen real

Como se puede apreciar el ruido de sal y pimienta de intensidad 0.1 perjudica de gran manera la imagen inicial. Veamos los resultados de la red:



Figura 4.6: Ejemplo de imagen de entrenamiento junto a imagen real

En este caso, la red ha conseguido de manera casi perfecta solucionar el problema a pesar del grado de distorsión.

Para este problema, a partir de la 10^a época el ruido ya era prácticamente nulo, aun que si faltaba cierto grado de definición en los pequeños detalles.

Probablemente la red actuó tan bien debido a:

1. **Características locales:** Las redes convolucionales están diseñadas para capturar características locales en una imagen. El ruido de sal y pimienta generalmente afecta solo a píxeles individuales o a pequeñas vecindades, lo cual es una característica local. Las capas convolucionales de la red pueden detectar estas características locales y aprender a restaurar los píxeles ruidosos basándose en las características de los píxeles vecinos.
2. **Mapeo no lineal:** Las redes convolucionales utilizan funciones de activación no lineales, como la función ReLU (Rectified Linear Unit), que les permite modelar relaciones no lineales en los datos. Esto es beneficioso para corregir el ruido de sal y pimienta, ya que el ruido puede introducir cambios drásticos y no lineales en los valores de los píxeles, y la red puede aprender a manejar estos cambios de manera efectiva.

4.2.3. Transformación logarítmica

A continuación, una foto con la transformación logarítmica aplicada, y la original.

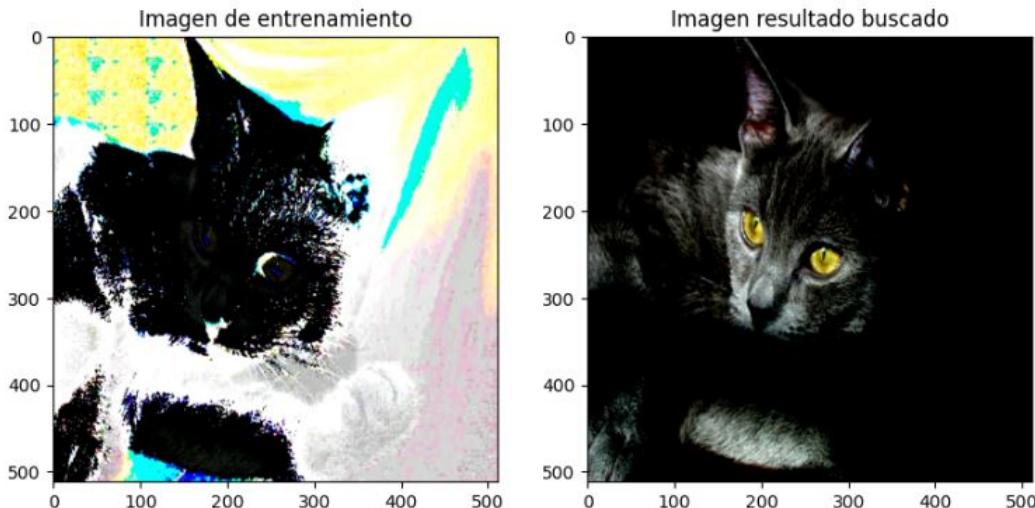


Figura 4.7: Visualización con Transformación de logaritmo

Al ejecutarse el modelo, podemos ver como la mejora de las imágenes va progresando adecuadamente conforme llega a la última época, mejorando la resolución en gran medida, difiriendo en varias características como la luz en algunas partes de la imagen o colores diferentes.

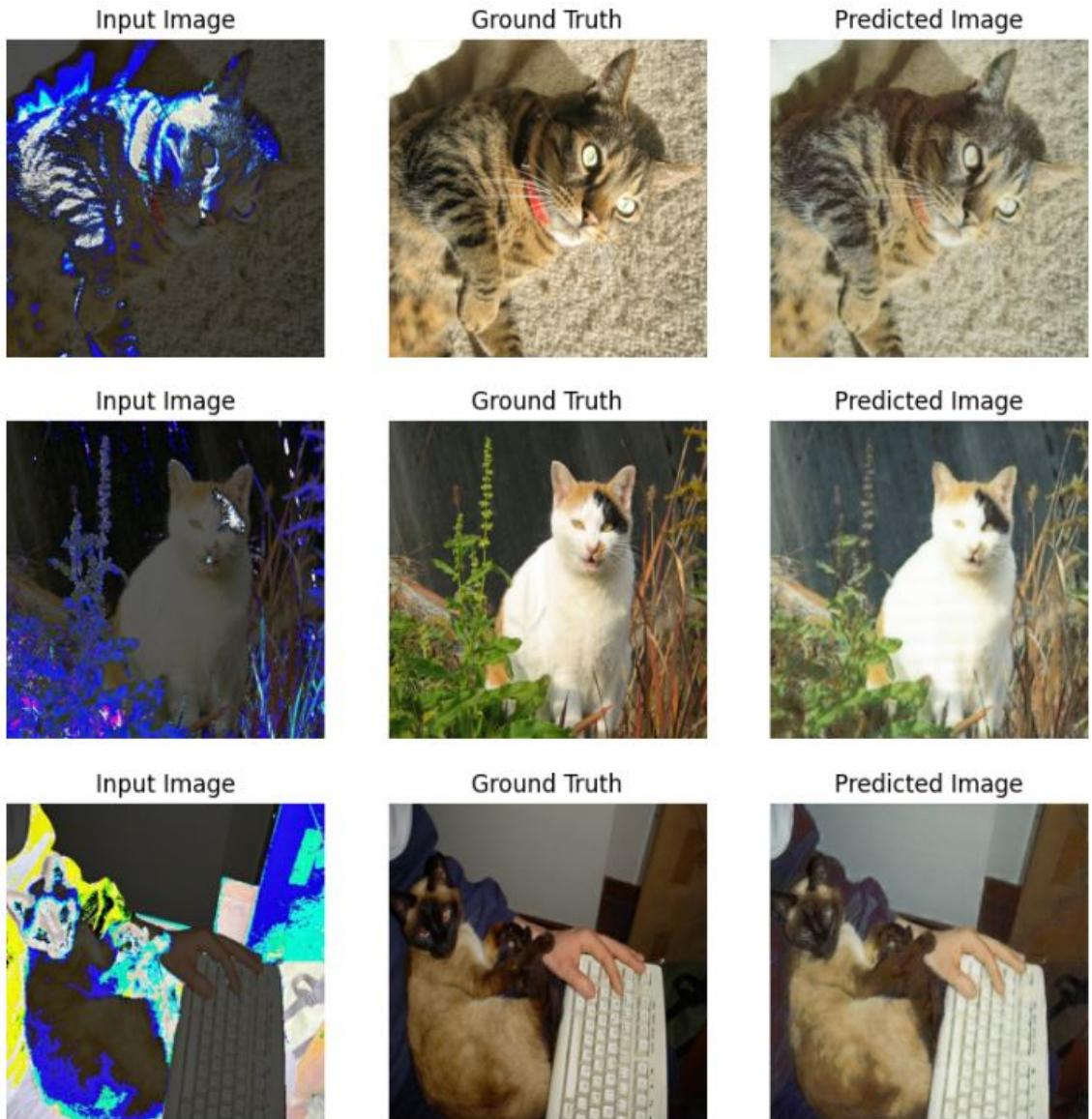


Figura 4.8: Resultados I

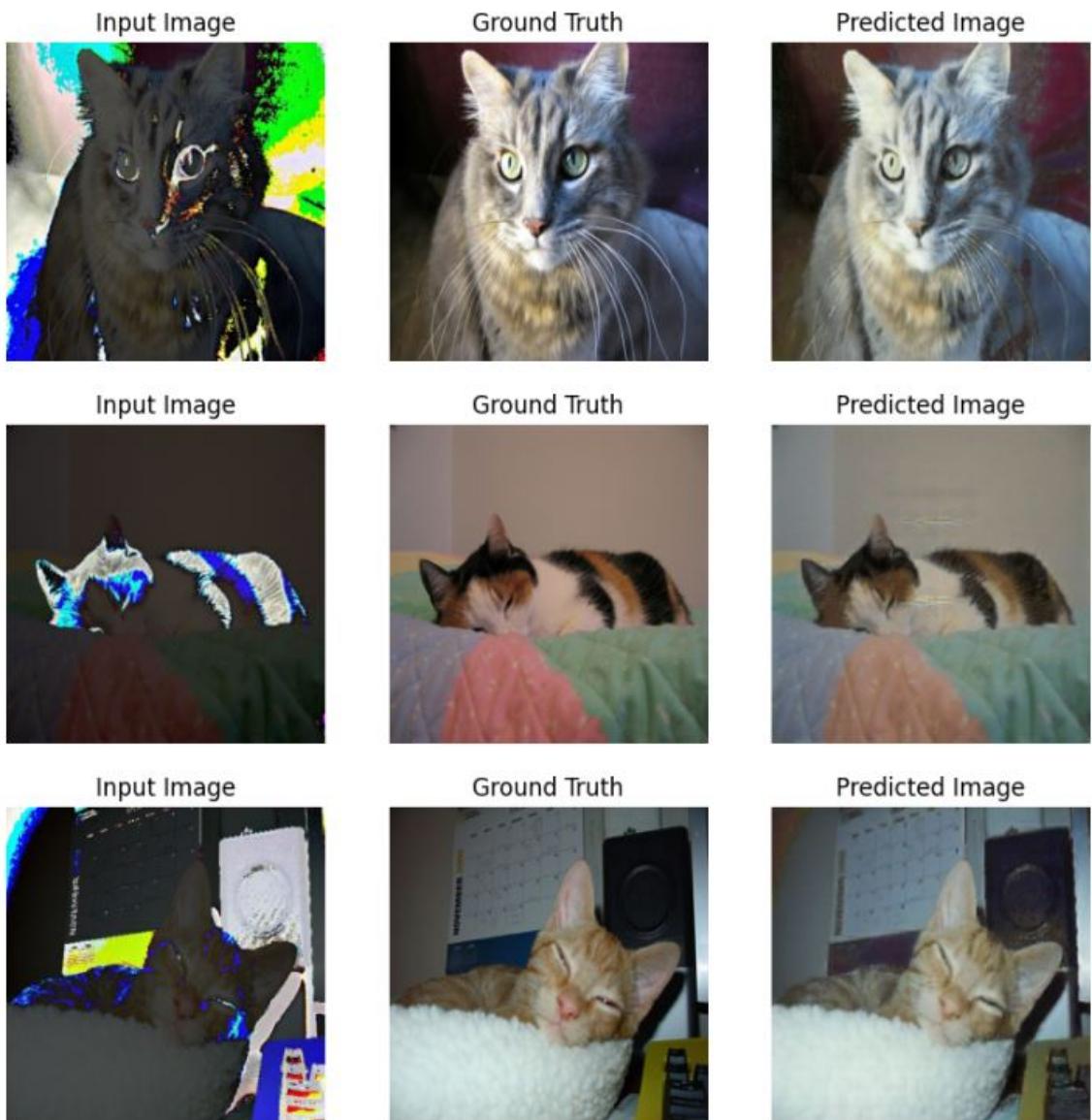


Figura 4.9: Resultados II

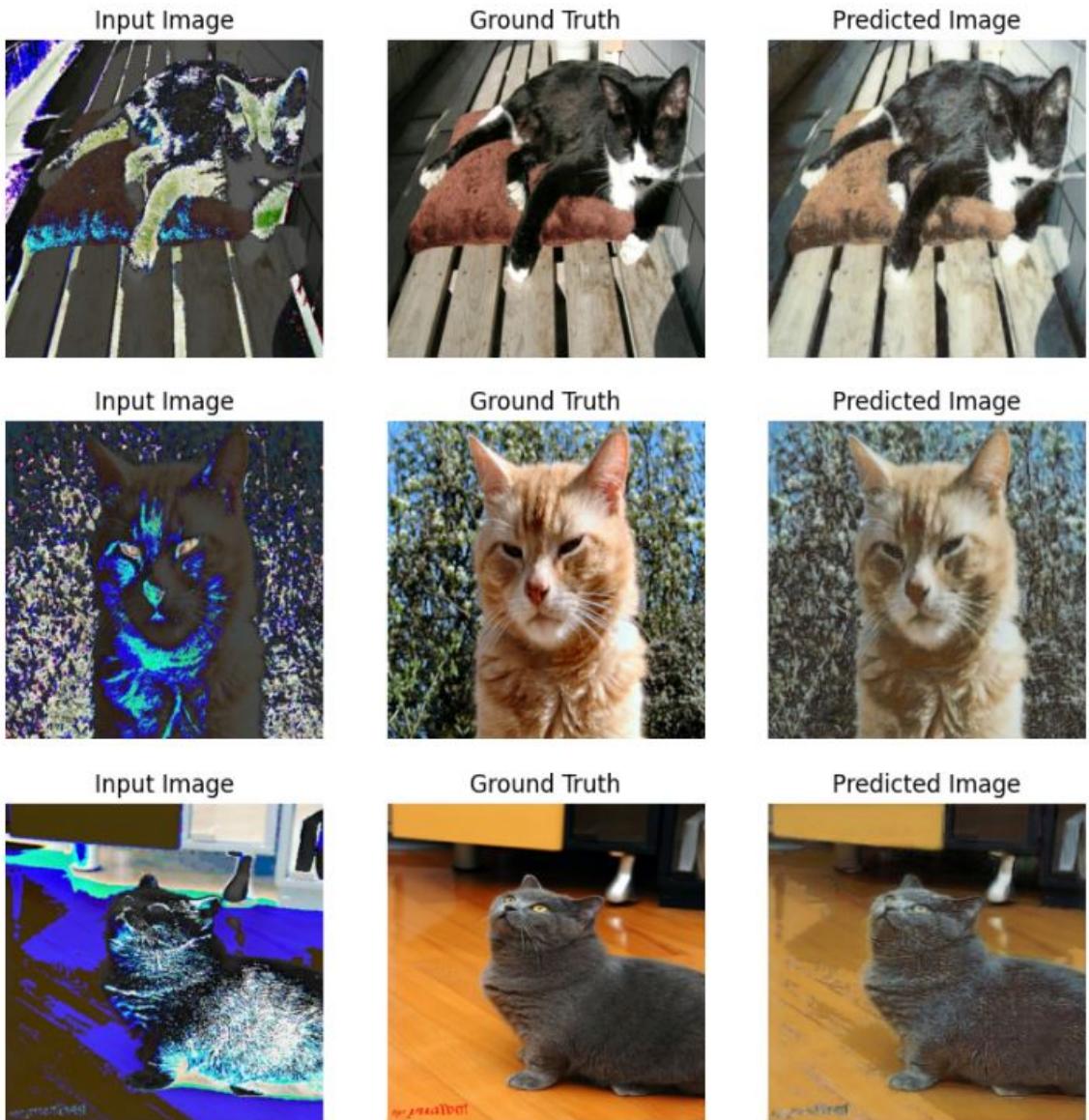


Figura 4.10: Resultados III

Esto puede ser debido a los siguientes factores:

1. **Perdida de información:** La transformación logarítmica nos ayuda a mejorar el contraste de las imágenes con baja intensidad. Debido a esto, se pueden producir pequeños errores debido a que los valores con alta intensidad pueden llegar a perderse o verse afectados, provocando así, que la red neuronal no sea capaz de representar dichos valores correctamente.
2. **Sobreentrenamiento:** Tras entrenar el modelo con 100 épocas, se pueden producir errores en algunas imágenes debido a que estas se están sobreentrenando lo que conlleva a una pérdida de calidad de imagen y en ocasiones, la aparición de líneas onduladas en algunas partes de esta.

4.2.4. Sobel

A continuación, se muestra una imagen aplicándole Sobel y su homónima original.

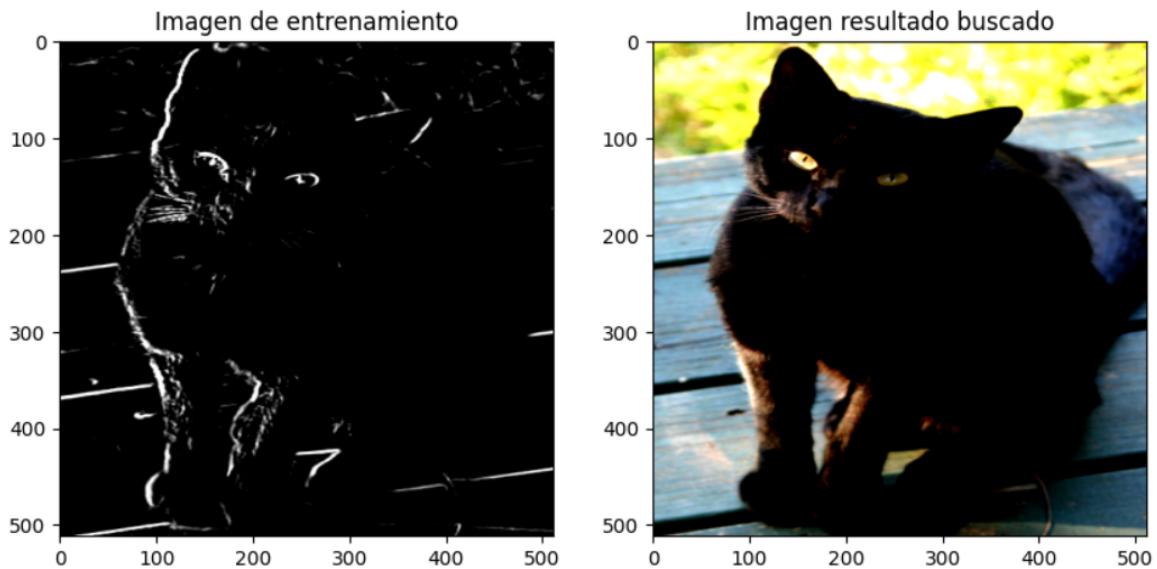


Figura 4.11: Visualización con Sobel

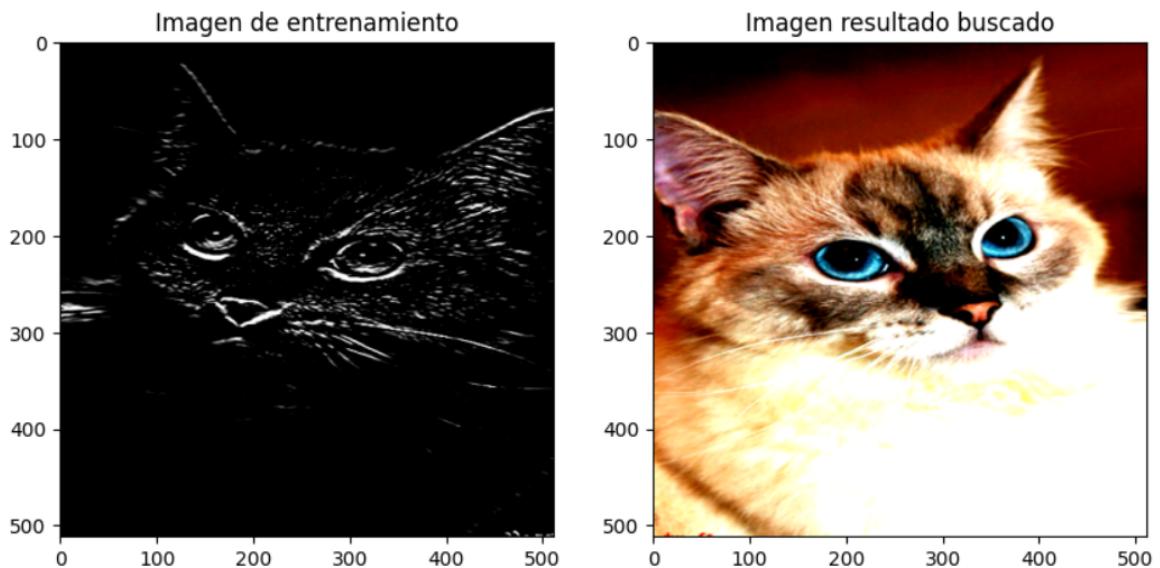


Figura 4.12: Visualización con Sobel con rotación

Como podemos ver en los siguientes ejemplos, a medida que la red neuronal mejora su predicción durante el entrenamiento, puede reconstruir, interpretando los colores y formas según los bordes, pudiéndose ver en la imagen final la silueta de los

gatos. Para aquellas imágenes que se tienen mas bordes (mas características), el modelo puede predecir y crear de una manera mas clara. Esto puede verse en las caras. Por lo demás no se le puede exigir mucho mas a este entrenamiento, teniendo en cuenta la dificultad de la imagen de entrada.

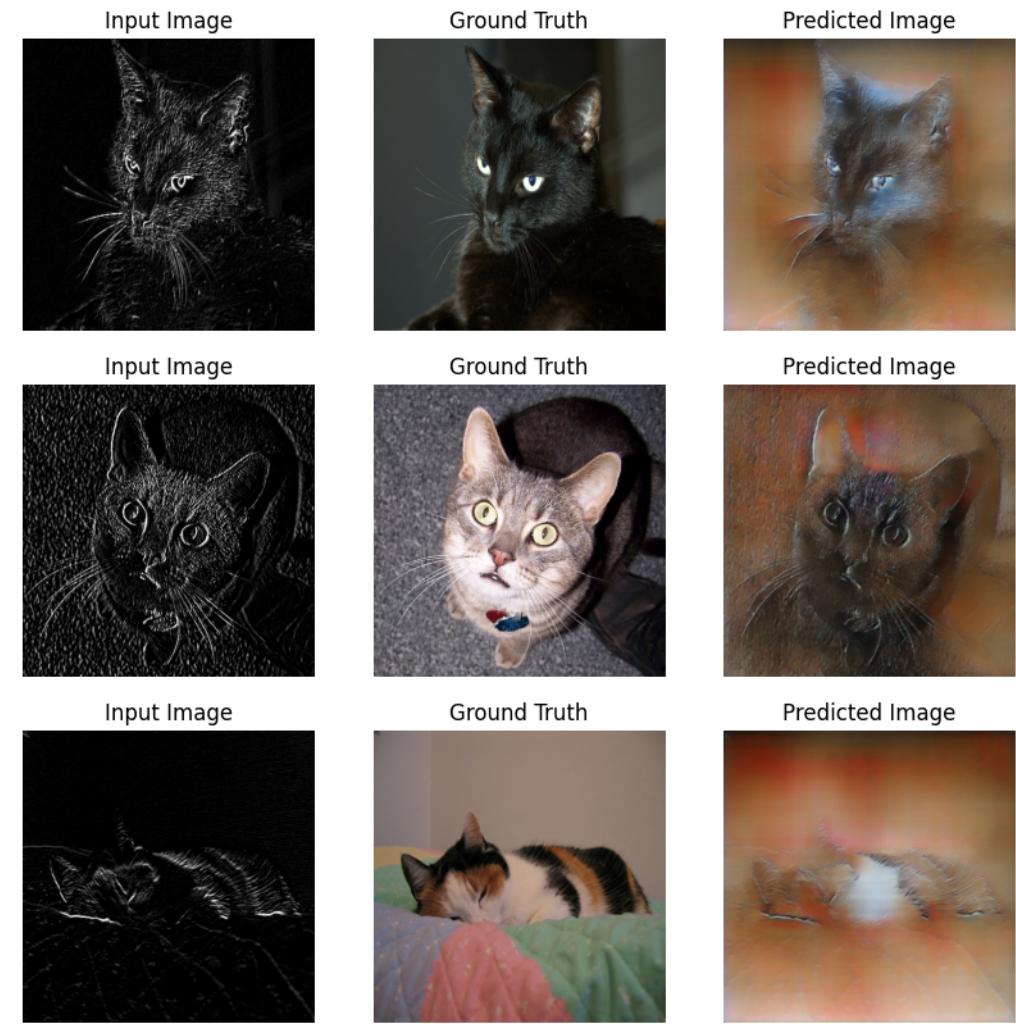


Figura 4.13: Resultados I



Figura 4.14: Resultados II

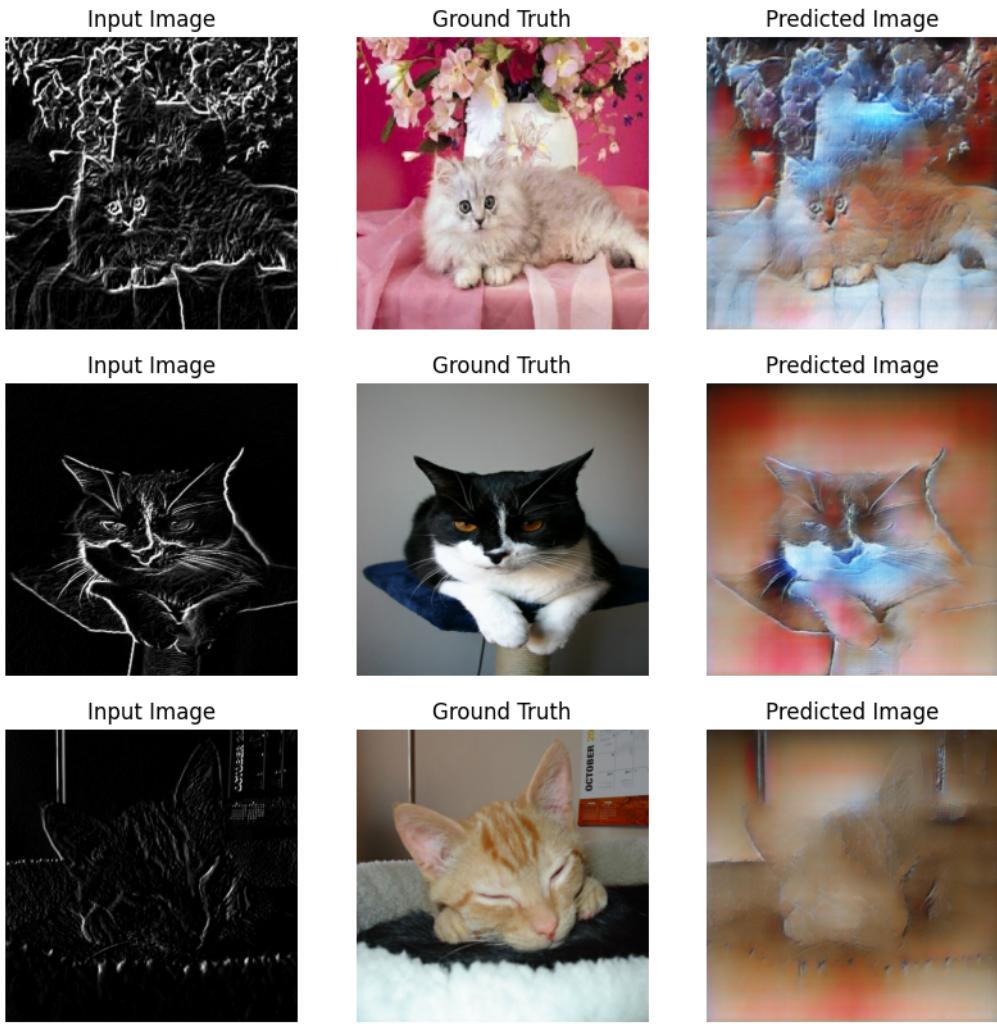


Figura 4.15: Resultados III

Sin embargo, existen algunos problemas y limitaciones que han podido afectar a la calidad de la reconstrucción de este apartado:

- **Información perdida:** El filtro Sobel se enfoca en enfatizar los bordes y las transiciones nítidas de la imagen, lo que significa pérdida de información de color y áreas suaves. Como resultado, la red neuronal tiene dificultades para restaurar los colores y texturas originales en ciertas áreas de la imagen.
- **Complejidad del modelo:** La arquitectura y las capacidades del modelo limita la capacidad de aprender las características y los patrones necesarios para reproducir con precisión la imagen original. A medida que el modelo aprende, es posible que pueda reproducir mejor algunos aspectos de la imagen, pero puede que no sea lo suficientemente complejo o no pueda capturar todos los detalles.
- **La calidad del conjunto de datos de capacitación:** El rendimiento del modelo también depende de la calidad y la cantidad de datos de capacitación disponibles. La colección de datos de capacitación es limitada y con imágenes donde el filtro no llega a captar bien el gato debido a que los bordes no son pronunciados

o la imagen es de mala calidad, cosa que hace que el modelo no pueda aprender a generalizar casos nuevos y diferentes.

4.3. Pruebas realizadas

Se decidió ejecutar una única prueba para cada método de preprocesamiento utilizando los mismos hiperparámetros. Las razones de esta decisión se exponen a continuación:

1. **Establecer un punto de partida común:** Al utilizar los mismos hiperparámetros en todas las pruebas, hemos establecido un punto de partida común para evaluar el impacto de cada método de pretratamiento en el rendimiento del modelo. Esto permite una comparación más directa y objetiva de los resultados, ya que cualquier cambio en el rendimiento se debe principalmente a las diferencias en el preprocesamiento de imágenes.
2. **Limitaciones de tiempo y recursos computacionales:** Probar a fondo diferentes combinaciones de hiperparámetros para cada método de preprocesamiento puede consumir mucho tiempo y recursos computacionales. El uso de un conjunto fijo de hiperparámetros nos ha permitido concentrarnos en evaluar el impacto de los métodos de preprocesamiento sin perder mas tiempo del necesario.
3. **Enfoque en la robustez del modelo:** Mientras se mantienen los hiperparámetros constantes, se puede comprobar el objetivo principal de las pruebas, que es evaluar la resistencia del modelo a varios métodos de preprocesamiento. Esta estrategia determina si el modelo puede adaptarse y generalizar en función de la calidad y las características de las imágenes de entrada, lo cual es una consideración importante en las aplicaciones del mundo real.
4. **Simplificación de la interpretación de resultados:** El uso de un conjunto fijo de hiperparámetros simplifica la interpretación de los resultados obtenidos, ya que reduce el número de variables que pueden afectar el desempeño del modelo. Esto facilita la identificación de los pros y los contras de cada técnica de preprocesamiento y determina cuál es más eficaz según el contexto de la aplicación.

5. Conclusiones

En primer lugar, podemos enumerar una serie de posibles mejoras para el proyecto:

1. Optimizar la arquitectura de la red: se puede mejorar en gran medida el rendimiento y la eficiencia de nuestra GAN. Experimentar con diferentes tamaños y estructuras de capas, así como con modelos GAN más avanzados, puede generar mejoras significativas en la calidad de las imágenes generadas
2. Normalización de clases: Puede ser una estrategia útil para acelerar el proceso de aprendizaje y mejorar la estabilidad general de la red. Técnicas como la normalización por lotes o la normalización de versiones pueden ser particularmente útiles para este propósito.
3. Optimización de hiperparámetros: Al probar y ajustar la tasa de aprendizaje, el tamaño del lote y la cantidad de épocas, pudimos refinar aún más el rendimiento de la red y la calidad de las imágenes generadas. Además, una vez que se crean las imágenes, podemos explorar diferentes técnicas de procesamiento posterior para mejorar aún más su calidad.
4. El uso de filtros antialiasing: Mejorar del contraste y técnicas similares pueden mejorar la nitidez y la claridad de la imagen resultante.
5. Usar la reducción sensorial: En combinación con la reducción del error cuadrático medio (MSE), puede resultar en una mejor calidad de imagen. Esta función tiene en cuenta aspectos que van más allá de las simples diferencias de píxeles, lo que puede dar como resultado una imagen mejorada que es más agradable para el ojo humano.
6. Aunque el dataset es bastante variado, sigue teniendo peculiaridades que hacen que el aprendizaje empeore. Un dataset mas homogéneo sería lo idóneo, por ejemplo con fondos más difuminados.

Como posibles extensiones de este trabajo, se pueden comentar la aplicación de nuestro modelo a diferentes regiones de la imagen. Expandirse a áreas como imágenes médicas, imágenes satelitales o bellas artes puede presentar desafíos únicos y brindar valiosas oportunidades de aprendizaje. Cada uno de estos dominios tiene características y necesidades específicas que pueden requerir que adaptemos y mejoremos nuestra red. Otra extensión valiosa de nuestro trabajo podría ser la introducción de métodos de aprendizaje por transferencia. Al tomar modelos previamente entrenados y adaptarlos para que se ajusten a nuestro problema particular de escalado de imágenes, pudimos mejorar tanto el rendimiento como la eficiencia de la GAN. El aprendizaje por transferencia puede permitirnos extraer información de grandes conjuntos de datos y aplicarla a nuestro caso específico, mejorando así la calidad de la imagen escalada. Además, será interesante estudiar la fiabilidad y seguridad de nuestra GAN frente a diferentes tipos de ataques, como ataques de adversarios. Este análisis nos ayudará a comprender mejor las limitaciones de la red y nos proporcionará una base para desarrollar estrategias para

mejorar la confiabilidad de la red. La investigación de seguridad de GAN es un campo en crecimiento y nuestra contribución puede ser sustancial. El desarrollo de una interfaz fácil de usar es otra posible extensión para hacer que nuestra GAN sea más accesible para los no expertos. Este software permitirá a los usuarios de nuestra red escalar sus propias imágenes sin tener que entender los detalles técnicos de su funcionamiento. Esta extensión puede afectar significativamente la disponibilidad y usabilidad de nuestra tecnología. Finalmente, podemos consultar varias métricas para medir la calidad de las imágenes generadas por nuestra GAN. Esto puede incluir indicadores objetivos como PSNR o SSIM, así como indicadores subjetivos basados en la percepción humana. Desarrollar e implementar estas métricas nos permitirá comprender mejor la calidad de las imágenes mejoradas y sentar una base sólida para futuras mejoras.

En conjuntos, estamos orgullosos de la colaboración y la habilidad técnica notables que hemos desarrollado durante la realización de nuestro proyecto de escalado de visualización de GAN. Hemos trabajado juntos de manera efectiva, manteniendo una comunicación abierta y clara. Nuestra competencia técnica es necesaria para este proyecto. Implementar y refinar una red neuronal GAN requiere una comprensión, al menos superficial, de la IA, el aprendizaje profundo y el procesamiento de imágenes. A través de pruebas continuas, resolución de problemas e iteraciones, pudimos experimentar y analizar un modelo GAN que puede escalar imágenes de manera eficiente. Durante el transcurso del proyecto, también demostramos una excelente capacidad de aprendizaje y adaptación.

En última instancia, el éxito de nuestro proyecto se debe en gran parte a una combinación de trabajo en equipo efectivo, sólidas capacidades técnicas y la voluntad de aprender y adaptarse continuamente.

6. Tabla de tiempos

| OBJETIVO | TAREA |
|------------|---|
| MARZO | |
| 23/03/2023 | Proyecto GitHub inicial |
| 27/03/2023 | Lectura de documentos y ejemplos |
| 30/03/2023 | Experimentación con diferentes tipos de redes y datasets |
| ABRIL | |
| 3/04/2023 | Migración de proyecto GitHub y primera versión de la red |
| 6/04/2023 | Prueba entorno GPU y configuración de red neuronal de Sobel |
| 10/04/2023 | Prueba entorno GPU y configuración de red neuronal de Gauss |
| 13/04/2023 | Prueba entorno GPU y configuración de red neuronal de Sal y pimienta |
| 17/04/2023 | Prueba entorno GPU y configuración de red neuronal de T.Logarítmica |
| 20/04/2023 | Ejecución de entornos y visualizado de resultados |
| MAYO | |
| 04/05/2023 | Redactar borrador Memoria |
| 08/05/2023 | Revisar puntos desde <i>Titulo</i> hasta <i>Resumen</i> |
| 15/05/2023 | Revisar puntos desde <i>Experimentación</i> hasta <i>Conclusiones</i> |
| 11/05/2023 | Revisar puntos desde <i>Planteamiento teórico e Implementación</i> |
| 18/05/2023 | Revisión global |
| 25/05/2023 | Presentación |

7. Bibliografía

- [1] Belén, M. G. (2023). Temario de la asignatura. https://ev.us.es/webapps/blackboard/content/listContent.jsp?course_id=_63848_1&content_id=_3591545_1.
- [2] Brain, G. (2015). Tensorflowapi. <https://www.tensorflow.org/>. Accessed: 2023-05-08.
- [3] Brownlee, J. (2019). A gentle introduction to pix2pix generative adversarial network. <https://machinelearningmastery.com/a-gentle-introduction-to-pix2pix-generative-adversarial-network/>. Accessed: 2023-05-04.
- [4] Crawford, A. (2019). Cat dataset. <https://www.kaggle.com/crawford/cat-dataset>. Accessed: 2023-05-08.
- [5] Fundation, P. S. (2023). Python. <https://www.python.org/>.
- [6] Google (2023). Google colab. <https://colab.research.google.com/?hl=es>.
- [7] Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2017). Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*.
- [8] Nvidia (2023). Nvidia cuda. <https://developer.nvidia.com/cuda-toolkit>.
- [9] Santana, C. (2017). Canal de divulgación sobre inteligencia artificial, tecnología, ciencia y futuro.
- [10] TensorFlow (2022). Pix2pix: Image-to-image translation with a conditional gan. <https://www.tensorflow.org/tutorials/generative/pix2pix?hl=es-419>.