



Resumen

El proyecto desarrolla una plataforma interactiva avanzada para la mejora de imágenes mediante aprendizaje profundo. Utiliza una arquitectura modular de backend, diseñada con técnicas de clean code y programación orientada a objetos, gestionada con Git para control de versiones. El frontend se implementó con Gradio, permitiendo una integración sencilla y flexible de nuevos modelos.

Para el procesamiento de imágenes, se realizó un testeo masivo de diferentes modelos de aprendizaje profundo, seleccionando los más adecuados y reentrenando uno con un dataset propio. La optimización de hiperparámetros se llevó a cabo mediante Weight&Bias, asegurando el mejor rendimiento posible del modelo. La plataforma permite tareas como la sustitución de fondo, balance de blancos y corrección de luz, utilizando CUDA para optimizar el rendimiento.

Una característica innovadora es la inclusión de un chatbot basado en LLM mediante la API LlamaAPI, que interpreta las peticiones del usuario y transforma estas solicitudes en comandos estructurados para aplicar los modelos correspondientes. Este chatbot mejora la accesibilidad y usabilidad de la plataforma.

Todo el código está documentado y tipado, facilitando su mantenimiento y comprensión. La integración de diversas tecnologías y herramientas ha permitido crear una solución robusta y eficiente para la mejora de imágenes, destacando la capacidad de gestionar grandes flujos de imágenes con alta calidad y precisión.



Resum

El projecte desenrotlla una plataforma interactiva avançada per a la millora d'imatges mitjançant aprenentatge profund. Utilitza una arquitectura modular de backend, dissenyada amb tècniques de clean code i programació orientada a objectes, gestionada amb Git per a control de versions. El frontend es va implementar amb Gradio, permetent una integració senzilla i flexible de nous models.

Per al processament d'imatges, es va realitzar un testatge massiu de diferents models d'aprenentatge profund, seleccionant els més adequats i reentrenant un amb un *dataset propi. L'optimització d'hiperparàmetres es va dur a terme mitjançant Weight&Bias, assegurant el millor rendiment possible del model. La plataforma permet tasques com la substitució de fons, balanç de blancs i correcció de llum, utilitzant CUDA per a optimitzar el rendiment.

Una característica innovadora és la inclusió d'un bot basat en LLM mitjançant la API LlamaAPI, que interpreta les peticions de l'usuari i transforma estes sol·licituds en comandos estructurats per a aplicar els models corresponents. Este bot millora l'accessibilitat i usabilitat de la plataforma.

Tot el codi està documentat i tipat, facilitant el seu manteniment i comprensió. La integració de diverses tecnologies i ferramentes ha permés crear una solució robusta i eficient per a la millora d'imatges, destacant la capacitat de gestionar grans fluxos d'imatges amb alta qualitat i precisió.



Abstract

The project develops an advanced interactive platform for image enhancement through deep learning. It uses a modular backend architecture, designed with clean code and object-oriented programming techniques, managed with Git for version control. The frontend was implemented with Gradio, allowing easy and flexible integration of new models.

For image processing, a massive testing of different deep learning models was carried out, selecting the most suitable ones and retraining one with its own dataset. Hyperparameter optimisation was carried out using Weight and Bias, ensuring the best possible model performance. The platform allows tasks such as background replacement, white balance and light correction, using CUDA to optimise performance.

An innovative feature is the inclusion of an LLM-based chatbot using the LlamaAPI API, which interprets user requests and transforms these requests into structured commands to apply the corresponding models. This chatbot improves the accessibility and usability of the platform.

All code is documented and typed, making it easy to maintain and understand. The integration of various technologies and tools has allowed the creation of a robust and efficient solution for image enhancement, highlighting the ability to manage large image streams with high quality and accuracy.

RESUMEN EJECUTIVO

La memoria del TFG del Grado en Tecnología Digital y Multimedia debe desarrollar en el texto los siguientes conceptos, debidamente justificados y discutidos, centrados en el ámbito de la tecnologías digitales y multimedia

CONCEPT (ABET)	CONCEPTO (traducción)	¿Cumple? (S/N)	¿Dónde? (páginas)
1. IDENTIFY:	1. IDENTIFICAR:		
1.1. Problem statement and opportunity	1.1. Planteamiento del problema y oportunidad	S	1
1.2. Constraints (standards, codes, needs, requirements & specifications)	1.2. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones)	S	2
1.3. Setting of goals	1.3. Establecimiento de objetivos	S	2
2. FORMULATE:	2. FORMULAR:		
2.1. Creative solution generation (analysis)	2.1. Generación de soluciones creativas (análisis)	S	3-8
2.2. Evaluation of multiple solutions and decision-making (synthesis)	2.2. Evaluación de múltiples soluciones y toma de decisiones (síntesis)		12-41
3. SOLVE:	3. RESOLVER:		
3.1. Fulfilment of goals	3.1. Evaluación del cumplimiento de objetivos	S	42
3.2. Overall impact and significance (contributions and practical recommendations)	3.2. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas)	S	42-44



Índice

I Memoria

Capítulo 1. Introducción y objetivos	1
1.1 Introducción y contexto	1
1.2 Objetivos	2
Capítulo 2. Metodología.....	3
2.1 Lenguaje de programación.....	3
2.2 Identificación y Selección de Modelos	3
2.2.1 Identificación de modelos.....	3
2.2.2 Proceso de selección de modelos.....	3
2.3 Gestión de proyecto y control de versiones	4
2.4 Estructura del proyecto	6
2.5 Implementación de modelos y optimización	7
2.6 Interfaz de usuario	7
Capítulo 3. Selección de tecnologías.....	9
3.1 Python	9
3.2 PyCharm	9
3.3 Git y GitHub	9
3.4 Poetry	9
3.5 Jupyter Notebooks	10
3.6 PyTorch.....	10
3.7 CUDA	10
3.8 JSON.....	10
3.9 API Calling	10
3.10 Gradio	11
3.11 Weight & Bias	11
Capítulo 4. Desarrollo de la plataforma	12
4.1 Fase 1: Selección e implantación de modelos	12
4.1.1 Selección de modelos	12
4.1.2 Implementación de los modelos	16
4.2 Fase 2: Implementación de LLM.....	19
4.2.1 Uso de Llama API	19
4.2.2 Function calling	20



4.2.3	Clase Llama	20
4.3	Fase 3: Creación de la interfaz gráfica de usuario (GUI)	22
4.3.1	Editor manual	22
4.3.2	Editor ChatBot	23
4.3.3	Aplicación de los modelos.....	24
4.4	Fase 4: Entrenamiento de modelo.....	26
4.4.1	Selección del modelo.....	26
4.4.2	Generación del dataset.....	27
4.4.3	Hiperparámetros y entrenamiento	30
4.4.4	Inferencia en notebook	37
4.4.5	Implementación del modelo	38
4.5	Fase 5: Documentación del código	39
4.5.1	Documentación.....	39
4.5.2	Tipado.....	40
4.5.3	Logging.....	40
Capítulo 5.	Conclusiones y propuesta de trabajo futuro	42
5.1	Conclusiones	42
5.2	Propuesta de trabajo futuro	43
Bibliografía.....		45

II Anexos

Capítulo 1. Introducción y objetivos

1.1 Introducción y contexto

Desde los inicios de la interacción humano-máquina, la representación y captura de imágenes y videos han sido pilares fundamentales en el desarrollo tecnológico. La fascinación por este mundo ha impulsado a muchas personas, incluyéndome a mí, a explorar constantemente nuevas herramientas y utilidades que mejoren y faciliten nuestras vidas cotidianas. La evolución de la tecnología de imágenes ha sido crucial en numerosos campos, desde la medicina hasta la seguridad y el entretenimiento.

En este contexto, surgió la idea de desarrollar una herramienta que abordara la necesidad tanto de empresas como de usuarios individuales de mejorar la calidad de sus imágenes de manera eficiente y efectiva. Esta idea se fundamentó en la observación de un vacío en el mercado y en la creencia en el potencial de una solución basada en aprendizaje profundo.

Identifiqué la necesidad de garantizar que todas las imágenes, independientemente de su origen, tuvieran una calidad mínima adecuada para su procesamiento, especialmente en el contexto de grandes volúmenes de imágenes. Esta necesidad se tradujo en la concepción de una herramienta que pudiera procesar imágenes de manera automatizada, mejorando su calidad y optimizando su utilidad para diversos propósitos.

Con este objetivo en mente, diseñé una serie de casos de uso que abarcaban diversas problemáticas comunes en el procesamiento de imágenes, como mala iluminación, desenfoque, imágenes borrosas y deformaciones por lentes. También consideré la necesidad de reemplazar cielos inconsistentes en fotos de edificios por cielos más uniformes y despejados, lo cual podría ser útil tanto para empresas inmobiliarias como para usuarios individuales que deseen mejorar sus fotos de manera rápida y sencilla.

Con esta información, tuve la libertad de investigar y probar diferentes modelos de aprendizaje profundo, así como de crear una herramienta fácil de usar para aplicar estos modelos. El aprendizaje profundo permite que modelos computacionales compuestos de múltiples capas de procesamiento aprendan de representaciones de datos con múltiples niveles de abstracción. Estos métodos han mejorado drásticamente el estado del arte en el reconocimiento del habla, el reconocimiento visual de objetos, la detección de objetos y muchos otros ámbitos.

El aprendizaje profundo descubre estructuras complejas en grandes conjuntos de datos utilizando el algoritmo de *backpropagation* para indicar cómo una máquina debe cambiar sus parámetros internos que se utilizan en el análisis de datos que se utilizan para calcular la representación en cada capa a partir de la representación en la capa anterior [1].

Estoy convencido de que esta herramienta tiene el potencial de ser una solución integral y valiosa para mejorar la calidad de las imágenes, tanto para empresas como para usuarios individuales.

1.2 Objetivos

El objetivo principal es crear una herramienta fácil e intuitiva para la empresa, en la que en unos pocos clics puedan ser procesados grandes volúmenes de imagen. Para llegar a este punto estableceremos otros objetivos previos para que el producto final cumpla con el requisito mencionado anteriormente.

- Encontrar los modelos *State-of-the-art* para las diferentes tareas

Estos modelos SOTA (State-of-the-art) son los mejores modelos que puedes usar en un tarea en específico, esto lo pueden conseguir basándose en la precisión, velocidad de inferencia y otras métricas relevantes [2]. Es por eso que debemos realizar una investigación exhaustiva probando y valorando los mejores modelos para cada una de las diferentes tareas de mejora de imagen.

- Optimizar los modelos y homogeneizarlos

Esta tarea, es la continuación de la anterior, ya que una vez elegidos habrá que tener en cuenta que como vamos a tener una plataforma multimodal lo importante será tener los mejores modelos pero que siempre se puedan adaptar a los requerimientos de CPU y GPU de los que dispondremos y puedan trabajar en un entorno donde el flujo de la imagen sea admitido por cada uno de ellos.

- Tiempo de inferencia y compatibilidad

Al tratarse de una plataforma con fines de productividad es muy importante que tenga un tiempo de inferencia que sea notablemente mayor al que tendrían si esta tarea se realizara manualmente. Además de ser compatible con los diferentes tipos de imagen con las que trabaja la empresa.

- Estructura de proyecto y flujo de trabajo

Este es uno de los mayores retos, que consistirá en concentrar todos estos modelos en una estructura de proyecto clara y comprensible, todo esto tratando de llevar un control del trabajo con herramienta de control de versiones.

- Interfaz y usabilidad

Brindar al usuario una experiencia fácil y agradable, que no tenga mucho ruido visual y que cumpla su tarea, en este apartado además se introducirá una alternativa más experimental en la que el usuario podrá mejorar las imágenes mediante un chatbot que gracias a un *Large Language Model* será capaz de seleccionar los modelos adecuados.

- Innovación y nuevas tecnologías

Conseguir usar las tecnologías más vanguardistas de cara a brindar el mejor resultado a la empresa. Esto es debido a la exponencial evolución del mundo de la Inteligencia Artificial y la aparición de infinitas opciones, en las que el reto será ponerlas en armonía.

Capítulo 2. Metodología

El presente proyecto se desarrolla con el objetivo de mejorar grandes volúmenes de imágenes utilizando modelos de *deep learning*. A continuación, se detalla la metodología que se seguirá para llevar a cabo este proyecto, desde la selección de modelos hasta la implementación final de una interfaz de usuario eficiente y amigable.

2.1 Lenguaje de programación

El proyecto se desarrollará utilizando Python, debido a que es ampliamente reconocido como el mejor lenguaje de programación para *deep learning*. En este caso trabajaremos en el marco de PyTorch.

Python se integra fácilmente con otros lenguajes y herramientas, permitiendo una flexibilidad esencial para la investigación y el desarrollo. Su sintaxis simple y legible no solo acelera el proceso de desarrollo, sino que también facilita la colaboración entre miembros del equipo, reduciendo la posibilidad de errores y mejorando la mantenibilidad del código.

2.2 Identificación y Selección de Modelos

2.2.1 Identificación de modelos

El primer paso en el proceso será identificar y seleccionar los modelos de *deep learning* adecuados para las siguientes tareas específicas:

- Ajustar balance de blancos
- Eliminar ruido
- Corregir desenfoque
- Corregir distorsión de lente (ojo de pez)
- Reemplazar cielo de imágenes
- Corregir baja exposición

2.2.2 Proceso de selección de modelos

Para ello, realizaremos una revisión exhaustiva de la literatura y nos basaremos en *benchmarks* disponibles en la plataforma Papers with Code para identificar los modelos *state-of-the-art* (SOTA) para cada una de estas tareas. Analizaremos artículos académicos, repositorios de código abierto, documentación oficial y los *benchmarks* proporcionados por Papers with Code, que nos permitirán comparar el rendimiento de diferentes modelos de manera objetiva.

Una vez identificados los modelos potenciales, procederemos a descargarlos desde repositorios de GitHub. Aunque muchas de estas implementaciones ofrecen demos online en plataformas como Hugging Face, nos centraremos en descargar y probar los modelos en nuestro entorno local. Esto es crucial, ya que necesitamos que los modelos funcionen

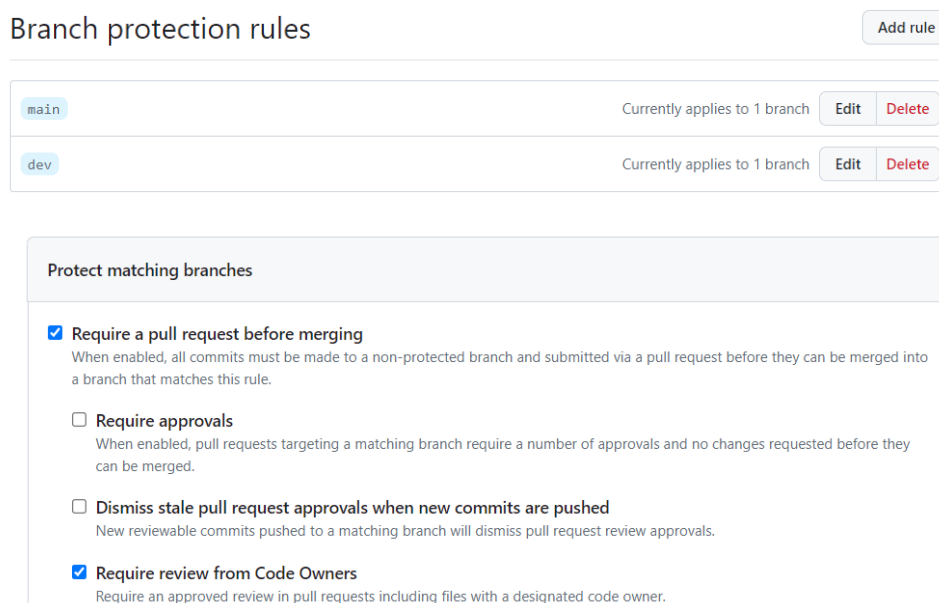
eficientemente en nuestra máquina para asegurar su viabilidad y optimización en el contexto específico del proyecto.

Evaluaremos el rendimiento de los modelos descargados en un entorno local, considerando la capacidad de ejecución en el hardware disponible, el tiempo de procesamiento y la precisión de los resultados. Implementaremos un enfoque iterativo de prueba y error para ajustar y optimizar los modelos seleccionados, realizando pruebas con conjuntos de datos específicos para cada tarea y evaluando métricas como la precisión, la velocidad de procesamiento y la robustez ante diferentes tipos de imágenes.

2.3 Gestión de proyecto y control de versiones

La gestión de este proyecto se llevará a cabo con Git y GitHub, asegurando un desarrollo organizado y colaborativo. Git ofrece ventajas significativas para la gestión de versiones y el trabajo en equipo. Permite el desarrollo paralelo mediante la creación de ramas separadas para cada nueva característica o corrección, lo cual facilita a los desarrolladores trabajar simultáneamente sin interferir con el código principal. Este enfoque ayuda a mantener un flujo de trabajo ordenado y permite la integración continua de nuevas funcionalidades que se han ido integrando en un repositorio [3].

Para asegurar la calidad y estabilidad del código, seguiremos prácticas estrictas en el manejo de ramas y fusiones. Tanto la rama “main” como la rama “dev” permanecerán protegidas de fusiones directas y será requerido un pull request previo, como se puede ver en la Figura 2.1. La rama “main” no se utilizará directamente hasta que las características estén completamente desarrolladas y probadas en la rama “dev”. Todo desarrollo nuevo se llevará a cabo en ramas separadas creadas desde “dev”, utilizando nombres descriptivos que indiquen claramente el propósito de la rama, como por ejemplo “feature/nombre-de-la-característica” [4].



The screenshot shows the 'Branch protection rules' interface in GitHub. At the top, there is a table with two rows: 'main' and 'dev'. Each row has a button to 'Add rule' and a status 'Currently applies to 1 branch'. Below the table, there is a section titled 'Protect matching branches' with several checkboxes and their descriptions:

- ☒ **Require a pull request before merging**
When enabled, all commits must be made to a non-protected branch and submitted via a pull request before they can be merged into a branch that matches this rule.
- ☐ **Require approvals**
When enabled, pull requests targeting a matching branch require a number of approvals and no changes requested before they can be merged.
- ☐ **Dismiss stale pull request approvals when new commits are pushed**
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.
- ☒ **Require review from Code Owners**
Require an approved review in pull requests including files with a designated code owner.

Figura 2.1. Configuración de la protección de las ramas main y dev

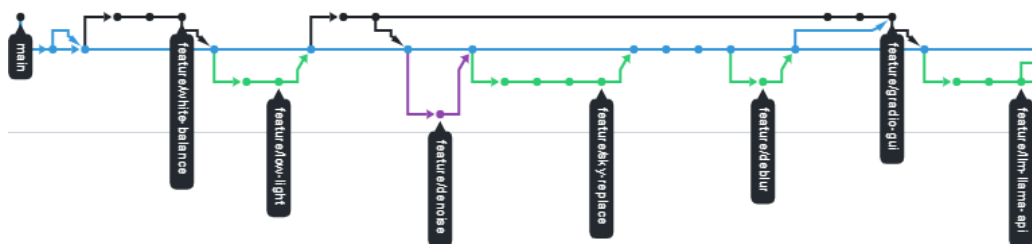


Figura 2.2. Timeline de commits y branches en GitHub

Cada cambio en el código se registrará con un historial detallado gracias a los commits de Git como se muestra en la Figura 2.3, lo que facilita el seguimiento de modificaciones y la reversión a versiones anteriores si es necesario. Además, se utilizarán pull requests para integrar cambios en “dev” y “main”, asegurando revisión y pruebas automáticas antes de la fusión, como se puede ver en la Figura 2.4. Esto no solo mejora la calidad del código, sino que también proporciona transparencia en el proceso de desarrollo.

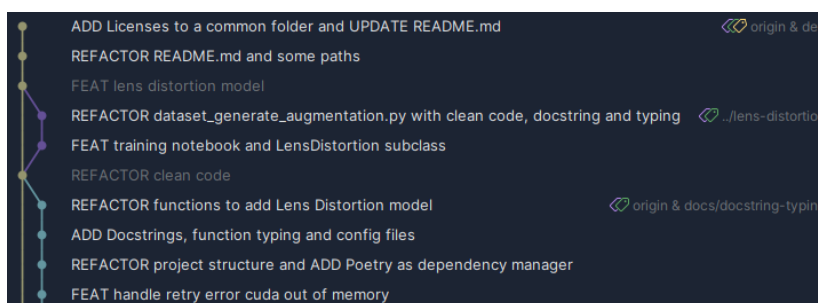


Figura 2.3. Historial de commits

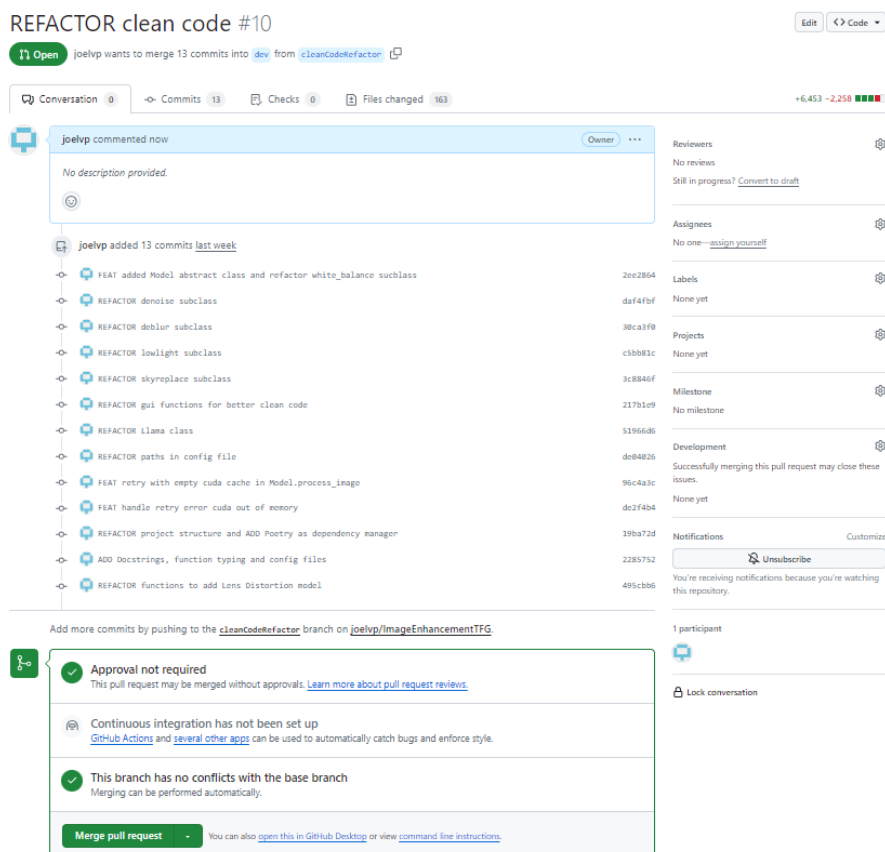


Figura 2.4. Pull Request previo a la fusión de ramas

El repositorio incluirá un README.md [5] detallado que proporcionará una guía clara sobre la estructura del proyecto, cómo configurarlo y ejecutarlo, se puede ver en la Figura 2.5, así como cualquier otra información relevante para colaboradores y usuarios. Este documento será una herramienta crucial para facilitar la adopción y comprensión del proyecto para cualquier persona que esté interesado en él.

Creación de Entorno Conda

Crea un nuevo entorno de Conda para gestionar las dependencias del proyecto:

```
conda create --name ImageEnhancementTFG python=3.10  
conda activate ImageEnhancementTFG
```



Instalación de Poetry y Librerías

Usaremos Poetry como gestor de dependencias:

```
pip install poetry  
poetry install
```



Configuración Adicional

Algunas configuraciones adicionales son necesarias para ciertos modelos. Navega al directorio `denoise_deblur` y ejecuta el siguiente comando:

```
cd imageenhancementtf/models/denoise_deblur  
python setup.py develop --no_cuda_ext
```



Figura 2.5. Explicación de instalación y ejecución en el repositorio de Git

En resumen, la combinación de Git y GitHub nos permitirá gestionar eficazmente el desarrollo del proyecto, mantener un código limpio y estable, y fomentar una colaboración fluida entre los miembros del equipo.

2.4 Estructura del proyecto

El proyecto se estructurará de manera modular utilizando PyCharm como entorno de desarrollo integrado (IDE). PyCharm ofrece potentes características de depuración, gestión de proyectos y soporte para Python, lo que facilita el desarrollo y la organización del código.

La estructura del proyecto será la siguiente:

- **imageenhancementtf:**
 - **data:** Esta carpeta contendrá archivos de configuraciones con datos sensibles como *keys*, los cuales no serán publicados en el repositorio remoto, además de archivos de configuración necesarios para el despliegue del programa.
 - **models:** Dentro de esta carpeta, se crearán subcarpetas para cada modelo de *deep learning* utilizado en el proyecto. Cada subcarpeta contendrá los scripts y archivos necesarios para el modelo específico.
 - **src:** Aquí se ubicará el objeto manejador de modelos, así como funciones auxiliares y otros scripts necesarios para la implementación y manipulación de los modelos.

- **gui.py**: Se trata del archivo que ejecutará la interfaz de usuario desarrollada con Gradio.
- **LICENSE**: Archivo de licencia del proyecto
- **poetry.lock**: Archivo de bloqueo de dependencias gestionado por Poetry.
- **Pyproject.toml**: Archivo de configuración del proyecto gestionado por Poetry.
- **README.md**: Archivo de documentación que proporciona una visión general del proyecto.

Esta estructura modular permitirá una gestión clara y eficiente del proyecto, facilitando la colaboración entre los miembros del equipo y asegurando una fácil mantenibilidad y escalabilidad del código.

2.5 Implementación de modelos y optimización

Cada modelo de deep learning se encapsulará en una clase de Python, permitiendo una gestión clara y modular de los modelos. Utilizaremos la programación orientada a objetos (OOP) para encapsular la lógica de cada modelo dentro de clases específicas. Esto permitirá una implementación más organizada y reutilizable, facilitando la integración y el mantenimiento de los modelos.

Se modificarán las funciones de los modelos para mejorar su rendimiento y eficiencia. Eliminaremos las funciones innecesarias para simplificar el código y reducir la complejidad. Implementaremos técnicas de optimización de hiperparámetros y ajustes finos para maximizar la precisión y eficiencia de los modelos. En algunos casos, se reentrenarán los modelos para ajustarlos más a los parámetros específicos del proyecto, utilizando conjuntos de datos personalizados y técnicas avanzadas de entrenamiento.

Para el reentrenamiento de modelos, utilizaremos Jupyter Notebooks, una herramienta interactiva que facilita el desarrollo y la experimentación con modelos de *deep learning*. Además, utilizaremos la plataforma Weights & Biases para monitorizar el proceso de entrenamiento, realizar un seguimiento de los experimentos y evaluar el rendimiento de los modelos ajustados. Weights & Biases proporcionará visualizaciones detalladas y métricas clave que nos ayudarán a tomar decisiones informadas durante el proceso de optimización.

2.6 Interfaz de usuario

Se desarrollará una interfaz de usuario utilizando Gradio, una herramienta ideal que ofrece una integración sencilla y directa con modelos de aprendizaje profundo, permitiendo la creación de interfaces interactivas que facilitan la experimentación y el despliegue de estos modelos. Sus características destacadas incluyen la facilidad para cargar y probar modelos, la capacidad de generar interfaces visuales atractivas y la

flexibilidad para manejar diferentes tipos de entradas y salidas. La interfaz tendrá dos secciones principales.

Una sección será la que denominemos “Manual”, los usuarios podrán seleccionar los modelos que desean utilizar para mejorar las imágenes. Proporcionaremos una interfaz intuitiva y directa, donde los usuarios podrán cargar sus imágenes y seleccionar las mejoras específicas que desean aplicar. Esta pestaña permitirá un control manual y detallado sobre el proceso de mejora de imágenes.

La segunda sección y más innovadora que denominaremos “ChatBot”, en esta sección, implementaremos un chatbot utilizando un modelo de lenguaje grande (LLM). Este chatbot utilizará técnicas de *prompting* y *function calling* para determinar automáticamente qué modelo se necesita en cada caso. Los usuarios podrán interactuar con el chatbot, describir las mejoras que desean, y el chatbot seleccionará y aplicará automáticamente los modelos adecuados, mejorando así la experiencia del usuario y facilitando la automatización del proceso.

Capítulo 3. Selección de tecnologías

En este capítulo, se detallarán las tecnologías seleccionadas para el desarrollo del proyecto, explicando sus funciones y la razón de su elección en el contexto específico del proyecto.

3.1 Python

Python es un lenguaje de programación de alto nivel conocido por su simplicidad y legibilidad. Es ampliamente utilizado en el desarrollo de aplicaciones de *machine learning* y *deep learning* debido a su vasta colección de bibliotecas y *frameworks* especializados, como TensorFlow, Keras y PyTorch.

Python se utilizará como el lenguaje principal para desarrollar todo el proyecto, desde la implementación de los modelos de *deep learning* hasta la creación de la interfaz de usuario y la gestión de datos. Su compatibilidad con múltiples bibliotecas y herramientas hace que sea ideal para integrar todas las funcionalidades necesarias de manera eficiente.

3.2 PyCharm

PyCharm es un entorno de desarrollo integrado (IDE) específico para Python, que ofrece características avanzadas como autocompletado de código, depuración y integración con sistemas de control de versiones.

PyCharm será utilizado para desarrollar y organizar el código del proyecto, además de para su ejecución. Sus funcionalidades avanzadas facilitarán la integración con el control de versiones Git, el manejo de librerías y la reparación de *bugs*.

3.3 Git y GitHub

Git es un sistema de control de versiones distribuido que permite rastrear cambios en el código y colaborar de manera eficiente, mientras que GitHub es una plataforma en línea que aloja repositorios Git y facilita la gestión de proyectos.

En nuestro proyecto, lo usaremos para gestionar el control de versiones, permitiendo un seguimiento detallado de los cambios y un manejo eficiente de ramas. Esto facilita el desarrollo ordenado, permite trabajar en nuevas funcionalidades sin afectar la rama principal y mantiene un historial claro de modificaciones, optimizando el flujo de trabajo individual.

3.4 Poetry

Poetry es una herramienta de gestión de dependencias y empaquetado para proyectos de Python, que simplifica la instalación y actualización de librerías, y asegura la consistencia del entorno de desarrollo.

En nuestro proyecto, utilizamos Poetry para gestionar las dependencias, asegurando que todas las librerías necesarias estén correctamente instaladas y actualizadas. Esto facilita la configuración del entorno de desarrollo y asegura la reproducibilidad del proyecto. Además, Poetry simplifica la instalación del proyecto para cualquier usuario que quiera

probarlo, garantizando que todas las dependencias se manejen de manera coherente y eficiente.

3.5 Jupyter Notebooks

Jupyter Notebooks es una aplicación web que permite crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto explicativo.

Jupyter Notebooks se utilizará para la experimentación y el desarrollo de prototipos de modelos de deep learning. Su capacidad para combinar código, visualización y documentación en un solo documento será invaluable para el análisis exploratorio de datos y el ajuste de modelos.

3.6 PyTorch

PyTorch es una biblioteca de *machine learning* de código abierto basada en la biblioteca Torch, que se utiliza principalmente para aplicaciones de *deep learning*. Permite cargar datasets, generar nuevos, aplicar transformaciones, reentrenar modelos etc.

PyTorch se utilizará para implementar y entrenar los modelos de *deep learning* necesarios para mejorar la calidad de las imágenes. Su flexibilidad y facilidad de uso lo hacen ideal para el desarrollo rápido y eficiente de modelos complejos.

3.7 CUDA

CUDA es una plataforma de computación paralela y un modelo de programación desarrollado por NVIDIA que permite utilizar la potencia de las GPUs para acelerar aplicaciones de cómputo intensivo, como el aprendizaje profundo.

En nuestro proyecto, utilizamos CUDA tanto para el entrenamiento de uno de nuestros modelos como para su ejecución y la de otros modelos. Esto permite aprovechar la aceleración por hardware de las GPUs, mejorando significativamente el rendimiento y reduciendo el tiempo de procesamiento. La integración de CUDA es esencial para manejar grandes volúmenes de datos y ejecutar modelos complejos de manera eficiente.

3.8 JSON

JSON (JavaScript Object Notation) es un formato de texto ligero para el intercambio de datos, fácil de leer y escribir tanto para humanos como para máquinas.

JSON se utilizará para almacenar datos y *prompts*, facilitando el intercambio de información entre diferentes componentes del proyecto. Su formato simple y estructurado será útil para manejar configuraciones, resultados y parámetros de los modelos.

3.9 API Calling

Las llamadas a API permiten la comunicación entre diferentes aplicaciones, facilitando el acceso y la integración de servicios externos.

Se utilizarán APIs como GoogleImageSearch para descargar fondos de imagen y aplicar al cielo, y LlamaAPI para llamar a modelos de LLM. Estas integraciones permitirán

enriquecer las funcionalidades del proyecto, ofreciendo soluciones avanzadas y personalizadas para la mejora de imágenes.

3.10 Gradio

Gradio es una herramienta que permite crear interfaces de usuario interactivas para cualquier modelo de *machine learning*, facilitando su despliegue y prueba.

Gradio se utilizará para desarrollar la interfaz de usuario del proyecto. Esta herramienta permitirá a los usuarios interactuar fácilmente con los modelos de *deep learning*, tanto en con la herramienta manual como con el chatbot con LLM.

3.11 Weight & Bias

Weight & Bias es una plataforma para la gestión de experimentos de *machine learning* que permite realizar un seguimiento del rendimiento de los modelos, comparar resultados y colaborar en proyectos.

Weight & Bias se utilizará para monitorear y registrar los experimentos de entrenamiento de los modelos, facilitando el seguimiento del rendimiento y el ajuste de hiperparámetros y por tanto elección del modelo final

Capítulo 4. Desarrollo de la plataforma

El desarrollo de la plataforma se dividirá en cinco fases principales:

4.1 Fase 1: Selección e implantación de modelos

4.1.1 Selección de modelos

En la primera fase del desarrollo de nuestra plataforma multimodal, nos centramos en la selección e implantación de modelos de *deep learning* adecuados para mejorar la calidad de imágenes en los siguientes aspectos; eliminación de ruido, corregir desenfoque, reemplazar el cielo, ajustar el balance de blancos y corrigiendo imágenes con baja exposición. Para ello, realizamos una exhaustiva investigación en el estado del arte de diferentes *benchmarks*, evaluando múltiples modelos hasta identificar los que presentaban el mejor rendimiento y precisión para nuestras necesidades. Todos los modelos seleccionados debían ser compatibles con PyTorch, garantizando así una integración fluida con el resto de nuestra plataforma.

Una vez identificados los modelos, pasamos a una fase intensiva de testeo en entornos de notebooks. Este paso fue crucial para asegurarnos de que cada modelo funcionaba correctamente en escenarios prácticos y que cumplía con los requisitos de rendimiento esperados. Durante esta etapa, realizamos pruebas con diversos conjuntos de datos y ajustamos parámetros para optimizar el comportamiento de los modelos. Este proceso nos permitió depurar posibles errores y verificar la robustez de cada modelo antes de su implementación definitiva en la plataforma. Los modelos seleccionados son los siguientes:

- **NAFNET**: eliminación de ruido y corrección de desenfoque.
Se trata de la implementación oficial del paper Simple Baselines for Image Restoration (ECCV2022) [6] [7]. Se caracteriza por su arquitectura innovadora que prescinde de funciones de activación no lineales, como ReLU o tanh, que son componentes comunes en las redes neuronales convolucionales (CNN) tradicionales. A diferencia de las CNN convencionales, NAFNet implementa activaciones lineales y módulos de atención para procesar información. Esta arquitectura única le permite lograr un rendimiento superior en diversas tareas de restauración de imágenes.
La creación de su arquitectura se basó en tres fases:
 1. Base robusta: arquitectura en forma de U con conexiones a salto para reducir la complejidad.
 2. Optimización: normalización por capas, activación GELU y atención por canal para mejorar el rendimiento.
 3. Simplificación: eliminación de activaciones no lineales con SimpleGate y atención por canal simplificada.

Todo esto derivó en un resultado innovador que, sin activaciones no lineales, iguala o supera el rendimiento de métodos SOTA en restauración de imágenes.

Demostrando que la simplicidad y la eliminación de activaciones no lineales pueden ser beneficiosas para este tipo de redes. Esta evolución de una estructura de red U con conexiones residuales a una red robusta sin activaciones no lineales se puede observar en la siguiente Figura 4.1 [6]:

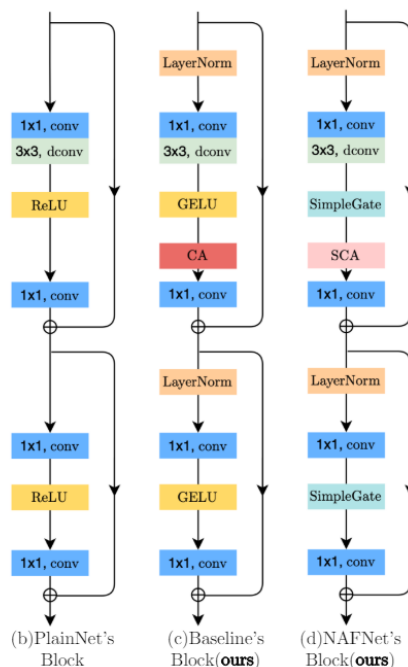


Figura 4.1. Evolución de la arquitectura de NAFNet. Fuente: [6]

La implementación de esta red neuronal en mi plataforma se resume a utilizar las tareas de eliminación de ruido y corrección del desenfoque, la estructura en su correspondiente carpeta se refleja en la Figura 4.8.

- **LLFlow:** Corregir baja exposición.

Se trata de la corrección de imágenes con baja exposición normalizando el flujo [8] [9]. Este método ofrece un enfoque novedoso que utiliza un modelo de flujo normalizador. Este modelo aprende la distribución compleja de las imágenes con luz natural y utiliza ese conocimiento para crear versiones mejoradas de las imágenes con poca luz que sean más realistas y visualmente atractivas.

Su arquitectura se divide en dos partes:

1. Codificador condicional: generar mapas de color invariantes de la iluminación, robustos y de alta calidad, las imágenes de entrada se procesan primero para extraer características útiles y las características extraídas se concatenan como parte de la entrada del codificador construido por Residual-in-Residual Dense Blocks (RRDB).
2. Red invertible: aprende una distribución de imágenes normalmente expuestas condicionada a una de baja iluminación, además pretende aprender una relación de uno a muchos, ya que la iluminación puede ser diversa para un mismo escenario.

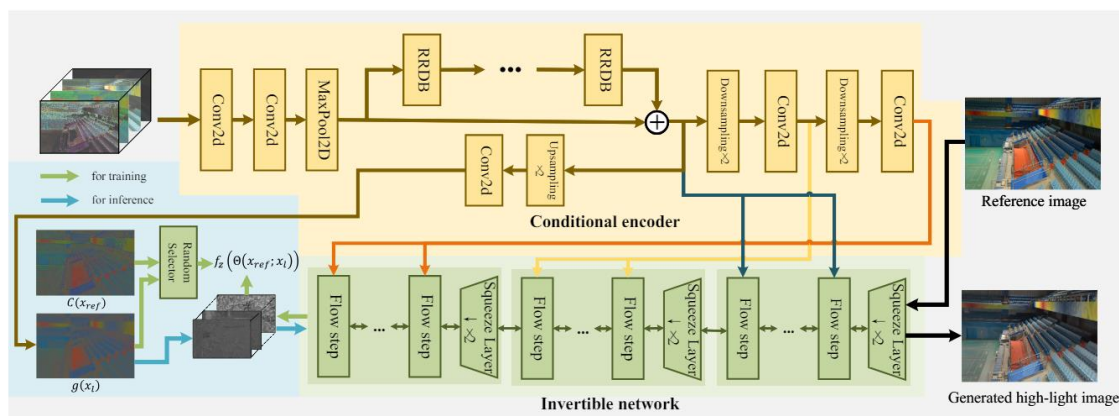


Figura 4.2. Arquitectura de LLEFlow. Fuente: [8]

La implementación de esta red neuronal en mi plataforma y la estructura en su correspondiente carpeta se refleja en la Figura 4.8.

- **Deep White Balance Editing:**

Se trata de la implementación oficial del paper Deep White-Balance Editing [10] [11]. Este modelo utiliza una red neuronal profunda para aprender a ajustar los tonos de color de una imagen de manera que se vean más naturales y realistas. Lo hace destacar entre otros modelos de corrección de color es su capacidad para generar múltiples opciones de balance de blancos para una misma imagen, mientras que otros modelos suelen ofrecer una única solución de corrección.

Esto lo consigue gracias a su arquitectura de red invertible, el modelo puede explorar diferentes combinaciones de colores y tonos, aprendiendo a mapear una imagen con bajo balance de blancos a múltiples imágenes con diferentes balances de blancos, todas ellas plausibles y visualmente atractivas.

Su arquitectura se divide en dos partes:

1. Codificador: analiza la imagen y extrae sus características más importantes, como los colores, texturas y condiciones de iluminación, convirtiéndola en un código numérico comprensible para la máquina. Este código actúa como una especie de resumen de la imagen original.
2. Múltiples decodificadores: El modelo emplea una arquitectura U-Net con múltiples decodificadores para generar diversas opciones de balance de blancos. Cada decodificador es esencialmente una red neuronal que reconstruye una imagen completa a partir del código generado por el codificador. Utilizando diferentes parámetros y estructuras en cada decodificador, el modelo puede producir múltiples resultados, ofreciendo una variedad de opciones de balance de blancos para una misma imagen.

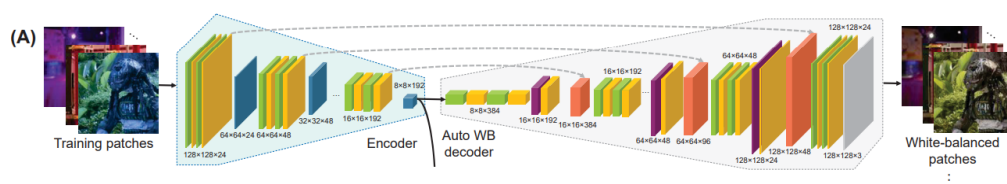


Figura 4.3. Arquitectura del modelo con AWB. Fuente: [10]

En nuestra casuística nos limitaremos a usar la red AWB (Automatic White Balance), visible en la Figura 4.3., la estructura en su correspondiente carpeta se refleja en la Figura 4.8.

- **SkyAR:**

Se trata de la implementación oficial del paper Castle in the Sky: Dynamic Sky Replacement and Harmonization in Videos [12] [13]. Este modelo de aprendizaje profundo está diseñado para realizar un reemplazo dinámico de cielo en videos, integrando el nuevo cielo de forma natural y coherente con el resto de la escena. A diferencia de técnicas tradicionales, este método permite realizar cambios de cielo de manera más realista y flexible, adaptándose a las variaciones de iluminación y contenido de los videos.

Lo que hace destacar a este modelo es su capacidad para armonizar el nuevo cielo con la escena existente. Esto significa que no solo se reemplaza el cielo, sino que se ajustan los colores, la iluminación y las sombras para que el cielo integrado parezca parte de la escena original. Esto se logra gracias a una arquitectura de red neuronal diseñada específicamente para este tipo de tarea, que permite aprender las relaciones complejas entre los diferentes elementos de una imagen.

El sistema propuesto consta de tres módulos principales:

1. Red de segmentación de cielo: identifica y separa el cielo del resto de la imagen, generando una máscara suave que define la región del cielo.
2. Estimador del movimiento: calcula el movimiento del cielo en el video, asumiendo un movimiento afín para los objetos celestes.
3. Módulo de integración del cielo: combina el cielo deseado con la imagen original utilizando la máscara generada y los parámetros de movimiento. Además, ajusta la iluminación y color del cielo integrado para lograr un resultado más realista.

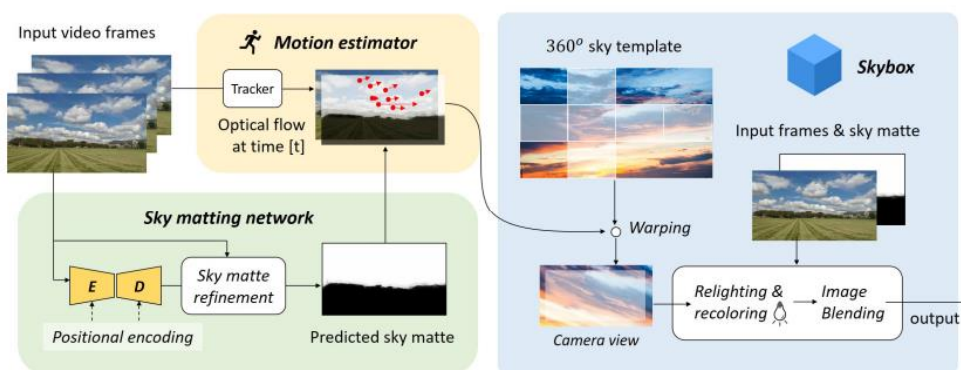


Figura 4.4. Arquitectura de SkyAR. Fuente: [12]

La implementación de esta red neuronal en mi plataforma es muy peculiar ya que pese a estar el modelo pensado para videos, realmente estos videos se dividen en frames y este preciso método es el que hemos extraído para implementar en nuestra plataforma, la estructura en su correspondiente carpeta se refleja en la Figura 4.8.

4.1.2 Implementación de los modelos

Para asegurar que la integración y el mantenimiento de los modelos sean lo más eficientes y escalables posible, hemos puesto un énfasis especial en la estructura del código. Desarrollamos una clase abstracta llamada *Model*, que se muestra en la Figura 4.5, que establece una estructura común y modulable para todos los modelos de *deep learning* que se integren en la plataforma. Esta clase define métodos esenciales que deben ser implementados por todas las subclases, garantizando una coherencia en la carga y procesamiento de imágenes. Adicionalmente, realizamos una simplificación intensiva del código fuente, adaptando funciones específicamente para nuestros inputs y outputs, eliminando código innecesario e incluso actualizando y aplicando principios de *clean code*. Esta estructura modular no solo facilita la integración de nuevos modelos en el futuro, sino que también simplifica el proceso de mantenimiento y actualización de los modelos existentes.

```

13 class Model(ABC):
14     """Abstract base class for deep learning models..."""
15     @abstractmethod
16     def __init__(self, model_path: Optional[str] = None, config_path: Optional[str] = None, device: str = 'cuda'):
17         """Constructs all the necessary attributes for the model object..."""
18         self.device = torch.device('cuda' if device.lower() == 'cuda' and torch.cuda.is_available() else 'cpu')
19         logging.info(f'Using device {self.device}')
20         self.model_path = model_path
21         self.config_path = config_path
22         self.model = None
23
24     @abstractmethod
25     def load_model(self) -> None:
26         """..."""
27         pass
28
29     @abstractmethod
30     def process_image(self, input_image: np.ndarray) -> np.ndarray:
31         """Process an input image using the model with retry logic..."""
32         image = self._process_image_impl(input_image)
33         reset_gradio_flag()
34         return image
35
36     @abstractmethod
37     def _process_image_impl(self, input_image: np.ndarray) -> np.ndarray:
38         """Process an image. This method must be implemented by subclasses..."""
39         pass

```

Figura 4.5. Clase abstracta Model

Para ilustrar cómo se implementa un modelo específico siguiendo nuestra estructura modular, en la Figura 4.6, se muestra la subclase `WhiteBalance`, que hereda de `Model` y se encarga de mejorar el balance de blancos en las imágenes. Como se puede observar se implementa un método `load_model` en la que será obligatoria cargar la red neuronal en la variable `self.model` de la subclase. El otro método obligatorio de implementar se trata de `_process_image_impl` el cual será obligatorio que reciba una imagen en formato `np.ndarray` y devuelva la imagen mejorada en el mismo formato. Dentro de cada uno de estos métodos de cada subclase se ha implementado la lógica específica de cada modelo, conservando únicamente las funcionalidades necesarias para nuestra casuística y necesidades.

```

14 class WhiteBalance(Model):
15     """Subclass of Model for improving white balance in images..."""
16     def __init__(self, model_path=config['models']['wb_model'], device='cuda'):
17         """Constructs all the necessary attributes for the WhiteBalance object..."""
18         super().__init__(model_path=model_path, device=device, config_path=None)
19
20     def load_model(self) -> None:
21         """..."""
22         # Load only AWB model
23         awb_path = os.path.join(self.model_path, 'net_awb.pth')
24         if os.path.exists(awb_path):
25             net_awb = self._load_awb_model(awb_path)
26         elif os.path.exists(os.path.join(self.model_path, 'net.pth')):
27             net_awb = self._load_split_model(os.path.join(self.model_path, 'net.pth'))
28         else:
29             raise Exception('Model not found!!')
30         self.model = net_awb
31
32     def _process_image_impl(self, input_image: np.ndarray, task: str = 'AWB', maxsize: int = 656) -> np.ndarray:
33         """Process an input image to improve its white balance and return the enhanced image..."""
34         image = Image.fromarray(np.uint8(input_image))
35         out_awb = deep_wb(image, task=task.lower(), net_awb=self.model, device=self.device, s=maxsize)
36         return (out_awb * 255).astype(np.uint8)

```

Figura 4.6. Implementación de la clase abstracta Model en WhiteBalance

La implementación del modelo de reemplazo de cielo, representado por la clase SkyReplace, ofrece una solución avanzada y flexible para modificar cielos en imágenes.

Una de las diferencias más notables en esta implementación es que la función `_process_image_impl` infringe ligeramente los parámetros de entrada establecidos en la clase base. A diferencia de otros modelos que procesan solo una imagen de entrada, SkyReplace requiere dos imágenes: la imagen principal que se va a procesar y una imagen de fondo que reemplazará al cielo. Esta modificación es esencial, ya que el modelo necesita la imagen de fondo para realizar un reemplazo convincente del cielo en la imagen original.

Además, el modelo SkyAR estaba originalmente diseñado para trabajar con secuencias de video, aplicando el reemplazo de cielo cuadro por cuadro. En lugar de utilizarlo directamente para videos, se adaptó su función de procesamiento para que pudiera aplicarse a imágenes fijas. Esto se logró al reutilizar la función `synthesize` que sintetizaba el reemplazo de cielo frame a frame, pero aplicándola a imágenes individuales en lugar de secuencias. Esta adaptación permite que el modelo funcione eficazmente en contextos de fotografía, manteniendo la precisión y el realismo en el reemplazo del cielo.

El flujo de trabajo de este modelo comienza estableciendo el tamaño de salida en función de las dimensiones de la imagen de entrada, para luego realizar una serie de transformaciones y ajustes a la imagen a través de la red generativa (`net_G`). Posteriormente, se emplea un refinamiento de la máscara de cielo y un *blending* final para integrar de manera suave el nuevo cielo con la imagen original.

```
@retry(stop=stop_after_attempt(3), before=clear_cuda_cache)
def process_image(self, input_image: np.ndarray, background_image: np.ndarray = None) -> np.ndarray:
    """Process an input image and replace the sky with the given background image (optional)..."""
    image = self._process_image_impl(input_image, background_image)
    reset_gradio_flag()
    return image

Codeium: Refactor | Explain | X
1 usage  joelyp
def _process_image_impl(self, input_image: np.ndarray, background_image: np.ndarray) -> np.ndarray:
    """Implementation method for processing the input image and performing sky replacement..."""
    self.set_output_size(input_image)

    self.config.out_size_w = self.out_size_w
    self.config.out_size_h = self.out_size_h

    self.skyboxengine = SkyBox(self.config, background_image)

    img_HD = self.cvtColor_and_resize(input_image)
    img_HD_prev = img_HD

    syneth = self.synthesize(img_HD, img_HD_prev)
    syneth = np.array(255.0 * syneth, dtype=np.uint8)

    return syneth
```

Figura 4.7. Implementación de la clase abstracta Model y adaptación a SkyReplace

Finalmente, la estructura del proyecto con los modelos implementados se refleja en la Figura 4.8.



Figura 4.8. Estructura por carpetas del proyecto

4.2 Fase 2: Implementación de LLM

En el ámbito de la inteligencia artificial, los Modelos de Lenguaje Generativo (GenAI) y los Large Language Models (LLM) han revolucionado la interacción entre humanos y computadoras, permitiendo aplicaciones avanzadas en el procesamiento del lenguaje natural (NLP). Este proyecto se centra en el uso innovador de dos modelos específicos, mixtral-8x22b-instruct y llama3-8b, para crear un chatbot capaz de mejorar imágenes según las instrucciones del usuario. La solución destaca por su enfoque creativo, donde el LLM no realiza directamente las mejoras de las imágenes, sino que orquesta la secuencia de tareas a realizar.

Los Modelos de Lenguaje Generativo (GenAI) son una subcategoría de la inteligencia artificial diseñados para generar contenido, ya sea texto, imágenes, música u otros tipos de datos, a partir de datos de entrada. Estos modelos aprenden patrones a partir de grandes volúmenes de datos y son capaces de generar respuestas coherentes y contextualmente relevantes.

Los Large Language Models (LLM) son una clase específica de modelos de lenguaje generativo que se caracterizan por tener un gran número de parámetros. Estos modelos, como ChatGPT, Gemini, Mistral o LLaMA, en la Figura 4.9, son entrenados en vastas cantidades de texto para aprender las complejidades del lenguaje humano. Los LLM pueden realizar una amplia gama de tareas, desde traducción y resumen hasta generación de texto creativo y respuesta a preguntas e imágenes. La competitividad y la mejora de estos modelos ha sido exponencial en los últimos meses brindando tanto a empresas como a particulares la posibilidad de mejorar en diversas tareas que anteriormente necesitaban un tiempo mucho mayor.

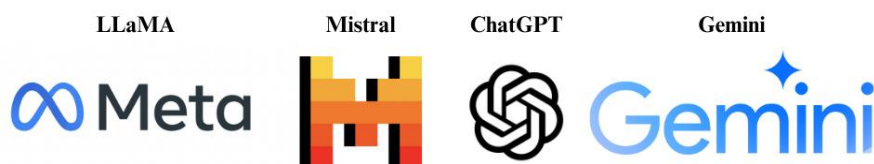


Figura 4.9. Principales modelos LLM

4.2.1 Uso de Llama API

La innovación principal radica en cómo el chatbot interpreta y responde a las entradas del usuario, diferenciándose según si el usuario proporciona solo una imagen, imagen y texto, o solo texto. Para esto usamos la librería de Python, Llama API, esto nos permitirá la interacción con modelos de lenguaje avanzados, facilitando la generación de respuestas

y la ejecución de tareas específicas basadas en los inputs del usuario. Soporta la interacción con una variedad de modelos de lenguaje, incluyendo mixtral-8x22b-instruct y llama3-8b, lo que permite aprovechar las capacidades avanzadas de estos modelos para diferentes tareas.

Una de las ventajas más significativas de esta solución creativa es la optimización de recursos. Los modelos multimodales de mejora de imágenes basados en LLM son costosos y requieren una gran cantidad de recursos computacionales para operar. En contraste, este enfoque analiza solo el texto, que es mucho más barato y rápido de procesar. Una vez que las tareas específicas se han identificado mediante la extracción de información del texto, las imágenes se mejoran utilizando redes neuronales especializadas implementadas localmente.

4.2.2 *Function calling*

El *Function Calling* es una característica innovadora introducida en los modelos de lenguaje recientes, como los de OpenAI, y en este caso la he aplicado con la librería de LlamaAPI. Su principal función es transformar la salida no estructurada de los modelos de lenguaje en datos estructurados, lo que permite un mayor control del output. Al invocar funciones, los modelos pueden devolver datos en formato JSON en lugar de texto natural, facilitando su uso en diversas aplicaciones, como en nuestra plataforma.

En el contexto del chatbot, el *Function Calling* desempeña un papel crucial. Aquí se utiliza para analizar el texto enviado por el usuario y extraer las tareas necesarias para mejorar una imagen, basándose en opciones predefinidas. Esta característica es especialmente importante ya que necesitamos una lista de *strings* válidos para poder llamar correctamente a los modelos de corrección de imágenes, en caso de no aplicar esta funcionalidad sería imposible integrar una LLM en una plataforma de estas características.

4.2.3 *Clase Llama*

La clase Llama es una implementación para interactuar con la API de Llama y generar respuestas basadas en entradas de texto e imagen. Esta clase es esencial para manejar diversas combinaciones de entradas y producir respuestas adecuadas mediante el uso de diferentes *prompts*.

La función principal es *generate*, en la Figura 4.10, encargada de generar una respuesta basada en el mensaje y la imagen proporcionados. Dependiendo de la combinación de entrada (solo texto, solo imagen, texto e imagen), se llama a diferentes métodos internos.

```
def generate(self, message: Optional[str], image: Optional[bool]) -> LlamaResponse:
    """Generate a response based on the message and image provided..."""
    if image and message:
        logging.info("Both image and text provided. Processing image and text input...")
        return self.handle_image_and_text(message)
    elif not image and message:
        logging.info("Only text provided. Processing text input...")
        return self.handle_text(message)
    elif image and not message:
        logging.info("Only image provided. Processing image input...")
        return self.handle_image_only()
```

Figura 4.10. Función principal en la generación de respuestas con Llama

- **Solo texto**

Para manejar la entrada cuando solo se proporciona texto se llama a la función *handle_text*. Utiliza el *prompt plain_text* para generar una respuesta adecuada, con este *prompt* lo que conseguiremos es que la LLM conteste a la pregunta o texto del usuario, pero siempre recordando que para lo que de verdad existe este chatbot es para mejorar imágenes. Para ello se especifica en la clave *system* además de brindarle un ejemplo de lo que sería una respuesta correcta.

- **Solo imagen**

Para manejar la entrada cuando solo se proporciona una imagen. Utiliza el *prompt only_picture* para generar una respuesta que elogie la imagen y solicite más información para mejorarla. De esta manera da la sensación que el chatbot está comprendiendo la imagen cuando realmente es una respuesta describiendo una imagen que no ha visto y pidiendo al usuario que le da la tarea necesaria para mejorarla.

- **Texto e imagen**

Este método maneja la entrada cuando se proporciona tanto una imagen como un texto. Utiliza un *prompt* específico *task_chooser* para extraer información relevante del mensaje y determinar las tareas necesarias para mejorar la imagen. Aquí es donde hay que tratar con más complejidad la entrada de texto ya que hay que extraer las tareas necesarias para mejorar la imagen, para ello aplicaremos el *function calling*. Esto nos permitirá convertir la salida desestructurada de la LLM a una salida concreta siguiendo nuestros parámetros, en nuestro caso devolverá un diccionario con las siguientes claves:

```
{'mejora_imagen': 'True', 'tareas': ['aumentando la luz'], 'cambio_cielo': 'True', 'new_background_image': 'sunrise wallpaper'}
```

Figura 4.11. Salida estructura de la LLM

Para obtener este resultado usamos el siguiente *prompt*:

```
"name": "extraccion-informacion",
"description": "Extrae la informacion relevante de la frase.",
"parameters": {
  "type": "object",
  "properties": {
    "mejora_imagen": {
      "title": "mejora_imagen",
      "type": "boolean",
      "enum": ["True", "False"],
      "description": "True si la frase indica que la imagen necesita una mejora, False si no necesita la mejora o aunque la necesite no especifique como."
    },
    "tareas": {
      "title": "tareas",
      "type": "List[strings]",
      "description": "Las tareas necesarias para mejorar la imagen. Opciones : \"quitando el ruido\", \"enfocando\", \"ajustando el balance de blancos\", \"aumentando la\",
    },
    "cambio_cielo": {
      "title": "cambio_cielo",
      "type": "boolean",
      "enum": ["True", "False"],
      "description": "True si la frase indica que se necesita cambiar el cielo/fondo de la imagen, False si no lo necesita"
    },
    "new_background_image": {
      "title": "new_background_image",
      "type": "string",
      "description": "El tema principal sobre el que debe ser el nuevo fondo de la imagen, incluye siempre 'wallpaper' y escríbelo en inglés, e.g. 'night sky with stars'"
    }
  }
}
```

Figura 4.12. Prompt para extraer las tareas de mejora de imagen con *function calling*

La función *generate* devuelve un objeto de tipo *LlamaResponse*, es una estructura de datos que encapsula la respuesta generada por la API. Utiliza el decorador `@dataclass` para simplificar su definición y la podemos observar en la Figura 4.13. Para poder alcanzar esta estructura todas las respuestas serán post procesadas y dependiendo de las condiciones que cumplan se creará de una manera o de otra.

```
@dataclass
class LlamaResponse:
    tasks: Optional[List[str]] = None
    new_background_image: Optional[str] = None
    response_text: Optional[str] = None
    error_message: Optional[str] = None
```

Figura 4.13. Objeto que devolverá la función *generate*

4.3 Fase 3: Creación de la interfaz gráfica de usuario (GUI)

La totalidad de la interfaz gráfica del usuario se ha desarrollado con Gradio, este *framework* ha sido elegido debido a su gran compatibilidad con modelos de inteligencia artificial, ofreciendo resultados en tiempo real y una estructura sencilla e intuitiva de cara al usuario. La interfaz se organiza en dos pestañas principales:

4.3.1 Editor manual

En la pestaña "Manual Editor", se ofrece una interfaz interactiva y eficiente para la mejora manual de imágenes. Los usuarios pueden cargar múltiples imágenes mediante un elemento de carga, que se visualizan en una galería organizada. A través de un menú desplegable, pueden seleccionar y ordenar los filtros a aplicar, incluyendo opciones como "Low Light", "Denoise", "Deblur", "White Balance", "Sky" y "Fish Eye". Si se selecciona el filtro "Sky", se habilita un campo adicional para cargar una imagen de cielo. Al presionar el botón de mejora "Enhance", se aplican los filtros seleccionados a las imágenes, cuyos resultados se muestran en una galería de salida. Muestro un ejemplo:

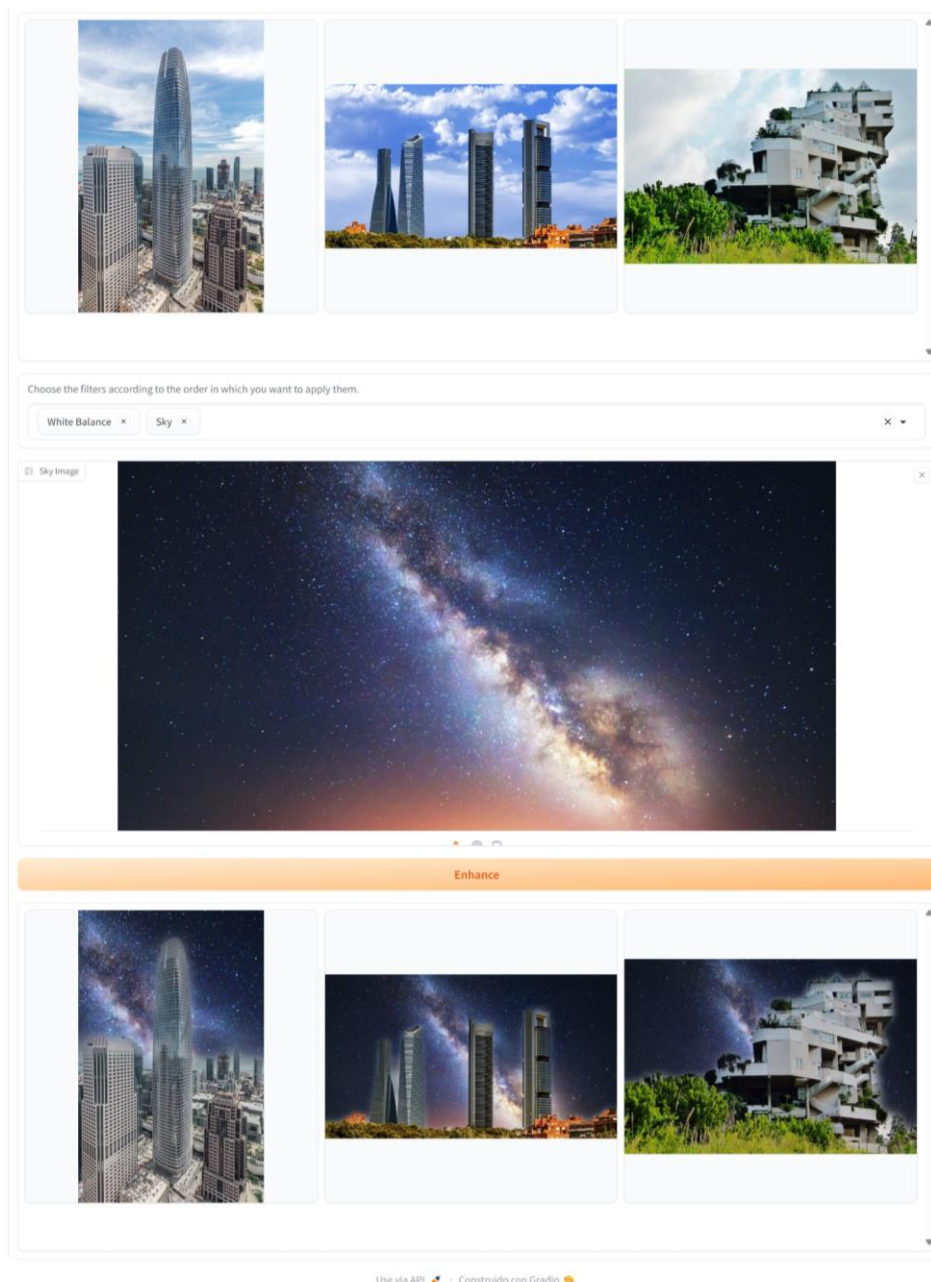


Figura 4.14. Interfaz de usuario con editor manual

4.3.2 Editor ChatBot

En la pestaña "ChatBot Editor", el componente principal es un chatbot diseñado para visualizar tanto el chat como las imágenes que los usuarios envían. Este chatbot permite una interacción en lenguaje natural para la mejora de imágenes, de manera que el chatbot entenderá de qué manera queremos mejorar la imagen que le enviamos.

Para la entrada de datos, se utiliza una MultimodalTextbox. Este elemento permite a los usuarios escribir mensajes y subir imágenes, recogiendo todas las entradas del usuario de manera eficiente. Los mensajes y las imágenes enviados se procesan y el chatbot genera la respuesta correspondiente, la cual se muestra en el mismo cuadro de chat.

El flujo de creado será capaz de diferenciar cuando el usuario ha añadido solo un mensaje, imagen y mensaje, solo imagen, solo mensaje, pero previamente en el chat había una imagen, etc. De manera que la LLM no solo mejorara imágenes, también es capaz de contestarte a cualquier otra pregunta, pero siempre te recordará que su tarea principal es mejorar imágenes. Esto se hace gracia a unos filtros y la posterior llamada al *prompt* oportuno de los explicados anteriormente. Muestro un ejemplo:

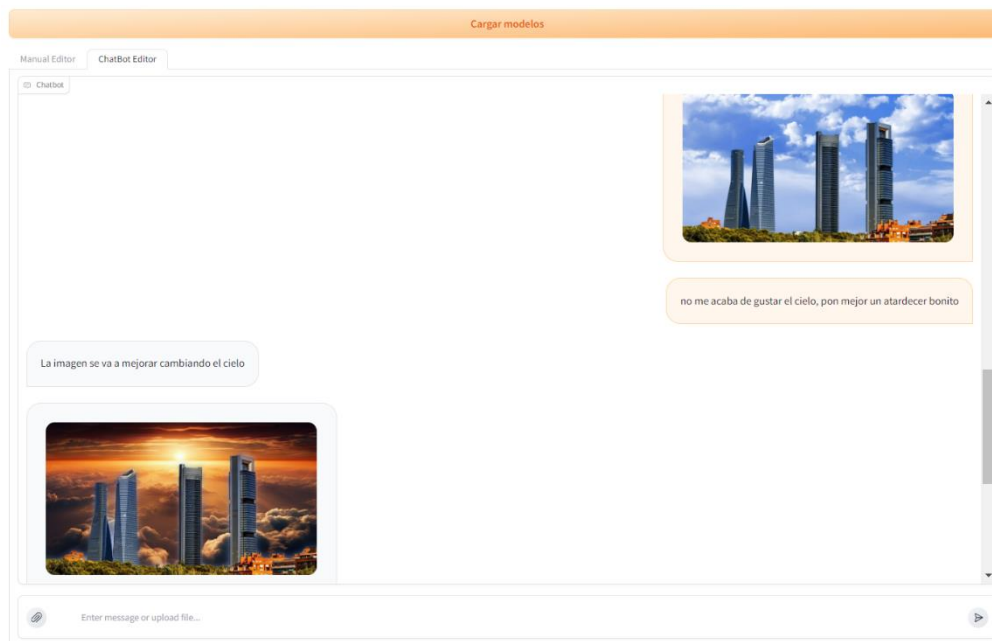


Figura 4.15. Interfaz de usuario con LLM ChatBot

4.3.3 Aplicación de los modelos

En la interfaz de la aplicación, tanto el Editor Manual como el ChatBot tienen como objetivo final la aplicación de transformaciones a las imágenes mediante la función *apply_transformations*. Aunque ambos enfoques permiten mejorar imágenes, el flujo de trabajo varía en cómo las imágenes y las opciones de filtro se recogen y procesan antes de llamar a esta función clave.

- Proceso en editor manual: Una vez seleccionadas las opciones de mejora de la imagen, al hacer clic en el botón "Enhance", se activa el evento *apply_transformations_event*. Esta función toma las imágenes cargadas y las opciones de filtro seleccionadas, llamando a la función *apply_transformations*. Dentro de *apply_transformations*, las imágenes y los filtros se pasan como parámetros junto con la imagen del cielo, si es necesario. Esta función en el archivo *utils.py* se encarga de cargar las imágenes desde las rutas proporcionadas, convertirlas al formato adecuado (RGB) y aplicar cada filtro seleccionado en el orden especificado. En caso de errores durante la aplicación de un filtro, estos se registran y se continúa con el siguiente filtro. Finalmente, las imágenes mejoradas se devuelven como una lista de matrices NumPy y se muestran en la galería de salida (*gr.Gallery*).

- Proceso en el ChatBot: Cuando el texto y las imágenes llegan a través del ChatBot, el flujo de trabajo es ligeramente diferente. Los usuarios interactúan con el chatbot enviando mensajes de texto o imágenes mediante el componente `gr.MultimodalTextbox`, y estos mensajes e imágenes se añaden al historial del chatbot. La función `add_message` actualiza este historial y desencadena la función `llm_bot`. En `llm_bot`, se analiza si el mensaje contiene texto, imágenes, o ambos, y dependiendo del contenido, se llama a `handle_image_and_text_input` si hay imágenes y texto. Esta función maneja las imágenes y el texto del mensaje, y si se requiere la mejora de las imágenes, llama a `apply_transformations` con las imágenes y las instrucciones del usuario. Aquí es donde se recoge la respuesta de Llama, que se forma como un objeto con posibles tareas e instrucciones. En el caso de que sea necesario cambiar el cielo en la imagen, se utiliza `google_image_search` para obtener una imagen adecuada del cielo, esta función usa la API de google images para elegir aleatoriamente entre las cinco primeras imágenes correspondientes a la búsqueda. En `apply_transformations`, similar al proceso del Editor Manual, las imágenes y las instrucciones se pasan para su procesamiento. Las imágenes se cargan desde las rutas proporcionadas, se convierten a formato RGB, y se aplican los filtros en el orden especificado. En caso de errores durante la aplicación de los filtros, estos se registran y se continúa con el siguiente filtro. Las imágenes mejoradas se devuelven como una lista de matrices NumPy y se añaden al historial del chatbot, mostrándose al usuario.

En resumen, tanto en el Editor Manual como en el ChatBot, la función `apply_transformations`, visible en la Figura 4.16., es el punto culminante del proceso de mejora de imágenes. Esta función es el punto de conexión entre el frontend y el backend, se encargará de recoger las tareas requeridas para cada imagen seleccionadas o interpretadas por la LLM en la interfaz y hacer las llamadas a las subclases que heredan de la clase abstracta `Model` y sus métodos. El archivo `utils.py` contiene una estructura bien organizada de funciones que aseguran un flujo de trabajo eficiente y manejan de manera efectiva las diversas entradas y operaciones requeridas, como pueden ser funciones para leer imágenes, para llamar a la API de google images, para transformar imágenes en *paths* temporales necesarios para mostrarlas en el chatbot etc. Esto permite un manejo fluido y adaptativo de las imágenes y opciones de mejora, proporcionando una experiencia de usuario optimizada y flexible.

```
def apply_transformations(input_images: List[str], options: List[str], model_manager: ModelManager,
    """Apply transformations to input images based on selected options..."""
    enhanced_images = []

    for image in input_images:
        image = imread(image)

        for option in options:
            try:
                if option == "Low Light":
                    if model_manager.ll_model is None:
                        model_manager.load_ll_model()

                    logging.info("Applying Low Light")
                    image = model_manager.ll_model.process_image(image)
                    logging.info("Low Light applied!")

                elif option == "Denoise":
                    if model_manager.denoise_model is None:
                        model_manager.load_denoise_model()

                    logging.info("Applying Denoising")
                    image = model_manager.denoise_model.process_image(image)
                    logging.info("Denoising applied!")
```

Figura 4.16. Fragmento de la función *apply_transformations*

4.4 Fase 4: Entrenamiento de modelo

4.4.1 Selección del modelo

En el desarrollo de este proyecto, se evaluaron diversos modelos de aprendizaje profundo para la corrección de distorsiones geométricas en imágenes, esta tarea era necesaria en el proyecto debido al gran volumen de fotografías tomadas con gran angular en dispositivos móviles o cámaras como la GoPro que se realizan a día de hoy. Entre las opciones consideradas, GeoProj se destacó como un candidato prometedor debido a su arquitectura específica para abordar este tipo de problemas.

Si bien GeoProj presenta ciertas limitaciones, como la escasez de documentación detallada [14] y un rendimiento inicial que requería mejoras, su potencial para ser adaptado y optimizado lo convirtió en la opción principal para este proyecto. Su flexibilidad, al permitir la modificación de su arquitectura y el entrenamiento con conjuntos de datos personalizados, lo posiciona como una herramienta poderosa para abordar las particularidades de nuestro dataset. Su enfoque en la corrección geométrica, sumado a la posibilidad de entrenarlo con un gran número de imágenes, lo convierte en una opción ideal para lograr los objetivos planteados en este trabajo.

Su arquitectura se divide en dos partes:

1. Extracción parámetros de distorsión: Inicialmente, se emplean tres capas convolucionales para capturar patrones visuales básicos como bordes y texturas. A continuación, se utilizan cinco bloques residuales para extraer características más complejas de manera gradual, preservando al mismo tiempo información detallada. Cada bloque residual consta de dos capas convolucionales con una

conexión de acceso directo (shortcut) que permite omitir el bloque, facilitando el flujo del gradiente durante el entrenamiento y mejorando el rendimiento de la red. Para reducir el costo computacional y centrarse en la información relevante, la imagen se reduce de tamaño mediante capas convolucionales con pasos más grandes (strides). Después de cada capa convolucional, se aplican funciones de normalización y activación ReLU para mejorar la estabilidad del entrenamiento y la capacidad de representación de la red. Finalmente, las características extraídas se aplanan en un vector unidimensional, que se introduce en una capa totalmente conectada para predecir los parámetros de distorsión [15].

2. Calculo del flujo: Estos parámetros, junto con un modelo de distorsión predefinido, que en nuestro caso solo usaremos el *Barrel*, generan el flujo de distorsión estimado. La red se entrena minimizando el error por píxel entre el flujo predicho y el flujo real.

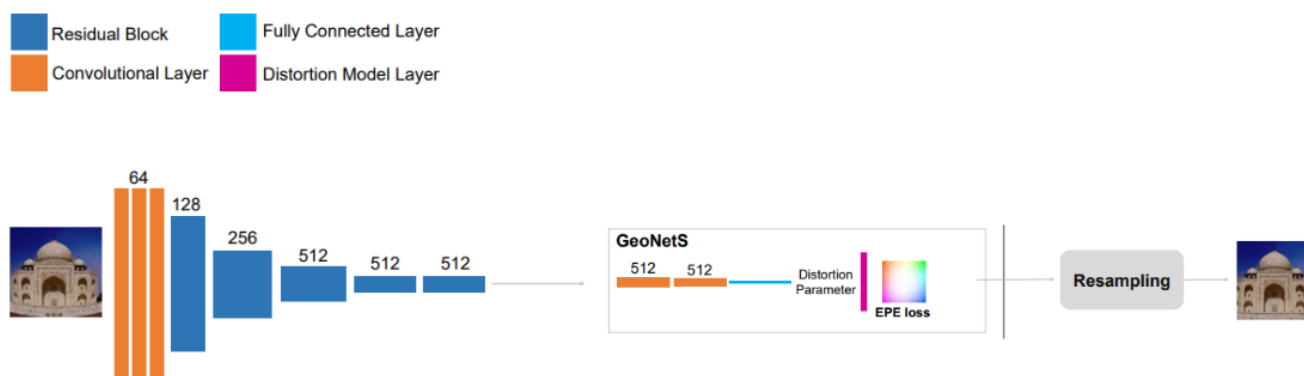


Figura 4.17. Arquitectura de GeoProj para la distorsión de lente (FishEye). Fuente: [15]

4.4.2 Generación del dataset

Para la generación del dataset primero había que partir de un dataset ya existente de imágenes normales, pero debía ser muy variado para que supiera comportarse con diferentes tipos de imágenes. Elegí los *datasets* Places365 [16] y un conjunto de fotografías tomadas por móviles debido a su diversidad y representatividad de escenas cotidianas. Esto proporcionaría una gran versatilidad, ya que abarca una amplia gama de situaciones y escenarios del día a día, aumentando así la robustez y generalización del modelo en situaciones reales.

Para crear este nuevo conjunto de datos, se desarrolló un script que sigue varios pasos clave. Primero, se configuraron los parámetros de entrada y se establecieron las transformaciones para la ampliación de datos, lo que ayuda a crear variabilidad en el conjunto de datos. El primer paso es filtrar las imágenes del conjunto de datos base, seleccionando aquellas que cumplieran con ciertos criterios de tamaño y características, para asegurar la calidad y consistencia del conjunto, esto se debe también a que si las imágenes no tienen un mínimo de alto y de ancho, exactamente el doble del tamaño de las imágenes del dataset, el proceso genera un error ya que no puede aplicar bien la distorsión

porque las distorsiones de lente, como la distorsión de barril, tienden a generar áreas "muertas" o bordes negros en los márgenes de las imágenes. Tener imágenes grandes permite recortar una sección central sin estos bordes indeseados, manteniendo así una imagen utilizable y centrada. En nuestro caso el dataset generado tendrá imágenes de 256x256.

En el siguiente paso del flujo de trabajo, se realiza la aumentación de datos para ampliar y diversificar el conjunto de imágenes disponible. Este proceso es fundamental para mejorar la capacidad del modelo de generalizar frente a nuevas imágenes. Se define una función *augment_data* que recibe como parámetros el número de imágenes aumentadas por cada imagen original y el tipo de transformación que aplicaremos. En este caso se aplica una transformación básica, con un volteo horizontal a la imagen con una probabilidad del 50%, esto introduce variabilidad en el dataset sin alterar la esencia de la imagen original y una rotación de la imagen al azar dentro de un rango de -45 a 45 grados que simula la captura de la misma escena desde diferentes ángulos y aumenta la robustez del modelo frente a rotaciones. Estas transformaciones permiten generar múltiples versiones de cada imagen original, introduciendo variaciones que simulan diferentes condiciones que el modelo podría encontrar en situaciones reales.

Para llevar a cabo la aumentación, se recorre cada imagen de la carpeta procesada y se le aplican las transformaciones definidas. De esta manera, por cada imagen original se crean varias nuevas, incrementando significativamente el tamaño y la diversidad del dataset.

```
transform_basic = T.Compose([
    T.RandomHorizontalFlip(p=0.5),
    T.RandomRotation(degrees=(-45, 45)),
])
```

Figura 4.18. Transformaciones aplicadas para la aumentación de datos

Después de realizar la aumentación de datos, el siguiente paso es organizar las nuevas imágenes generadas. Para ello, se utiliza una función que mezcla aleatoriamente las imágenes en la carpeta procesada y les asigna nuevos nombres. Este proceso de renombrado y mezcla es crucial para evitar que el modelo aprenda patrones no deseados basados en el orden o el nombre de los archivos.

A continuación, cada imagen recibe un nuevo nombre en un formato secuencial de seis dígitos, como "000001.jpg", "000002.jpg", y así sucesivamente, manteniendo siempre la extensión ".jpg". Este renombrado no solo ayuda a evitar conflictos de nombres que podrían haber surgido durante la aumentación, sino que también facilita la organización del conjunto de datos, asegurando que esté listo para el entrenamiento del modelo de manera eficiente y sin sesgos.

Para aplicar la distorsión, necesitaremos unos índices de comienzo y final tanto para el conjunto de entrenamiento como para el de prueba. La función *prepare_indexes_new_generation* se encarga de preparar estos índices cuando no existen imágenes generadas previamente. Primero, cuenta todas las imágenes disponibles en la

carpeta procesada, luego calcula cuántas se usarán para entrenamiento y cuántas para prueba, basándose en los porcentajes definidos. Finalmente, devuelve estos índices, con el conjunto de entrenamiento comenzando en 0 y el de prueba justo después. Esto asegura un punto de partida organizado para la generación de datos.

Para aplicar la distorsión a las imágenes, hay dos bucles que recorren cada imagen del conjunto de datos, primero imágenes para entrenamiento y luego de test. Primero, se calcula una región central de la imagen original y se aplica la distorsión pixel a pixel, ajustando las coordenadas según el modelo elegido. Luego, se recorta la imagen distorsionada para eliminar bordes negros, y se guarda en la carpeta correspondiente, dependiendo si es parte del conjunto de entrenamiento o prueba.

```
for k in range(trainnum_idx, trainnum_total):
    generatedata(types, folders, IMG_WIDTH_DATASET, IMG_HEIGHT_DATASET, k, train_flag=True)

for k in range(testnum_idx, testnum_total):
    generatedata(types, folders, IMG_WIDTH_DATASET, IMG_HEIGHT_DATASET, k, train_flag=False)
```

Figura 4.19. Bucle para aplicar distorsión al conjunto de imágenes

La generación de un dataset es un proceso largo y en el que es fácil que surjan fallos, por eso la función *prepare_indexes_started_generation* se diseñó para manejar interrupciones en la generación de imágenes distorsionadas, permitiendo que el proceso se reanude desde donde se había detenido. En lugar de comenzar desde cero, la función revisa las carpetas donde se almacenan las imágenes distorsionadas y los archivos de flujo correspondientes, tanto para el conjunto de entrenamiento como para el de prueba. Primero, calcula el número total de imágenes en el dataset procesado y determina cuántas deben destinarse a entrenamiento y cuántas a prueba. Luego, inspecciona las carpetas de imágenes distorsionadas y archivos de flujo. Si ya existen archivos, encuentra el índice más alto, asegurándose de que los índices de las imágenes y los archivos de flujo coincidan. Esto asegura que el proceso de distorsión no sobrescriba ni duplique imágenes y que, si la generación se detiene, pueda retomarse exactamente en el punto correcto, sin necesidad de repetir los pasos previos de aumento de datos o renombrado.

Todas estas funciones se van recogiendo y llamando desde el *main* del script *dataset_generate_augmentation.py*, como se muestra en la Figura 4.20., en mi caso un dataset inicial de 5500 imágenes, se convirtió en 22000 imágenes, con un factor de aumento de cuatro, en el que 20000 fueron destinadas para entrenamiento y 2000 para testeo.

```
##### MAIN #####
if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO)

    args = parser.parse_args()
    folders = create_folders(args)

    if not os.listdir(folders['processedFolder']): # If empty processed folder with dataset is new generation
        filter_images(folders, IMG_WIDTH_DATASET, IMG_HEIGHT_DATASET)

        ##### Augment Data #####
        if args.data_augmentation:
            augment_data(folders, args.augment_factor, transform_basic)

        ##### Rename and shuffle Images #####
        rename_and_shuffle_images(folders)

        ##### Prepare indexes #####
        trainnum_total, testnum_total, trainnum_idx, testnum_idx = prepare_indexes_new_generation(folders)

    else: # If processed folder with dataset exists, then check how many final images are there
        trainnum_total, testnum_total, trainnum_idx, testnum_idx = prepare_indexes_started_generation(folders, args)

    # Only barrel
    for types in ['barrel']:
        for k in range(trainnum_idx, trainnum_total):
            generatedata(types, folders, IMG_WIDTH_DATASET, IMG_HEIGHT_DATASET, k, train_flag=True)

        for k in range(testnum_idx, testnum_total):
            generatedata(types, folders, IMG_WIDTH_DATASET, IMG_HEIGHT_DATASET, k, train_flag=False)
```

Figura 4.20. Flujo de archivo *dataset_generate_augmentation.py*

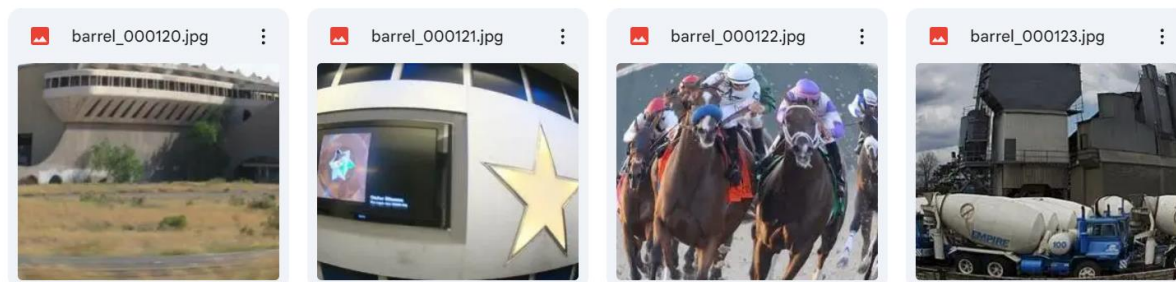


Figura 4.21. Imágenes con distorsión

4.4.3 Hiperparámetros y entrenamiento

4.4.3.1 Búsqueda de hiperparámetros

El proceso de búsqueda de hiperparámetros es crucial para optimizar el rendimiento de un modelo de aprendizaje profundo. Dado que los hiperparámetros, como la tasa de aprendizaje, el tamaño del lote, el optimizador y la cantidad de épocas, influyen significativamente en cómo aprende un modelo y en su capacidad para generalizar, es esencial encontrar la combinación óptima.

Para automatizar y hacer más eficiente esta tarea, he utilizado la herramienta Weights & Biases (W&B) junto con su funcionalidad de *sweeps*. En particular, la configuración del sweep está diseñada para encontrar la mejor combinación de hiperparámetros mediante un método bayesiano, que es efectivo para explorar el espacio de hiperparámetros de manera inteligente y enfocada, en lugar de probar combinaciones al azar.

La configuración que he utilizado incluye los siguientes elementos clave:

- **Método bayesiano:** Este enfoque busca minimizar la pérdida del modelo seleccionando de manera inteligente los próximos conjuntos de hiperparámetros en función de los resultados anteriores, lo que es más eficiente que una búsqueda aleatoria.
- **Término temprano con Hyperband:** Esta estrategia de terminación anticipada permite descartar configuraciones de hiperparámetros que no muestran potencial después de unas pocas iteraciones, ahorrando tiempo y recursos.
- **Batch size:** Se exploran tres tamaños diferentes (16, 32 y 64), lo que permite evaluar el impacto del tamaño del lote en la estabilidad y la velocidad del entrenamiento.
- **Learning rate:** Se busca en un rango continuo entre $1e-06$ y 0.001 , lo que permite identificar el mejor equilibrio entre la velocidad de aprendizaje y la estabilidad.
- **Optimizadores:** Se evalúan dos optimizadores populares, Adam y SGD, para encontrar cuál de ellos se adapta mejor a la tarea específica.

La métrica objetivo es minimizar la pérdida o *loss* del modelo, ya que es un indicador directo de cuán bien el modelo se ajusta a los datos de entrenamiento. Al usar un método de optimización bayesiano junto con el término temprano, es posible encontrar configuraciones de hiperparámetros que no solo resulten en un modelo preciso, sino que también maximicen la eficiencia del entrenamiento.

Este proceso es vital porque un buen ajuste de hiperparámetros puede ser la diferencia entre un modelo que tiene un rendimiento mediocre y uno que sobresale, además de ahorrar tiempo y recursos computacionales.

Las funciones *train* y *train_epoch* son el núcleo del proceso de entrenamiento del modelo, encargándose de todo, desde el manejo de los datos hasta la optimización del modelo y la evaluación de su desempeño.

La función *train_epoch* es la responsable de ejecutar el ciclo de entrenamiento para una sola época. Dentro de esta función, primero se inicializan las variables necesarias para acumular la pérdida a lo largo de los lotes de datos, lo que permite monitorear el rendimiento del modelo en detalle. Luego, se itera sobre todos los lotes de datos presentes en el *train_loader*. Cada lote contiene imágenes distorsionadas y sus correspondientes flujos de movimiento en las direcciones x e y, que se procesan para alimentar al modelo. Si se dispone de una GPU, los datos se transfieren a ella para aprovechar la aceleración en el cálculo. Durante la fase de *forward pass*, el modelo predice el flujo de movimiento para las imágenes de entrada a través de dos redes (*model_1* y *model_2*). Estas predicciones se comparan con las etiquetas reales utilizando una función de pérdida (*criterion*), que mide la discrepancia entre la predicción y el valor verdadero. Luego, mediante el *backward pass*, se calculan los gradientes y se ajustan los pesos del modelo a través del optimizador (*optimizer*). Este proceso se repite para cada lote de datos en la época. Al final de cada lote, la pérdida promedio se registra en W&B y se reinicia para la

siguiente iteración. Una vez procesados todos los lotes, se calcula y registra la pérdida promedio de la época completa, que es crucial para evaluar el progreso del modelo.

```
for i, (disimgs, disx, disy) in enumerate(train_loader):
    if torch.cuda.is_available():
        disimgs = disimgs.cuda()
        disx = disx.cuda()
        disy = disy.cuda()

    optimizer.zero_grad()

    labels_x = disx
    labels_y = disy

    flow_truth = torch.cat([labels_x, labels_y], dim=1)

    # Forward pass
    flow_output_1 = model_1(disimgs)
    flow_output = model_2(flow_output_1)

    # Calculate loss
    loss = criterion(flow_output, flow_truth)

    # Backward pass and optimization step
    loss.backward()
    optimizer.step()

    # Accumulate total loss
    cumu_loss += loss.item()
    batch_loss += loss.item()
```

Figura 4.22. Parte principal del entrenamiento en la función *train_epoch*

Por otro lado, la función *train* organiza y controla el proceso de entrenamiento completo, desde la configuración de los hiperparámetros hasta la evaluación del modelo en cada época. Se comienza inicializando un nuevo run en Weights & Biases, lo que permite rastrear todas las métricas y parámetros de esta sesión de entrenamiento. Luego, se cargan los hiperparámetros especificados por W&B, como el tamaño del lote, la tasa de aprendizaje, y el optimizador, los cuales determinarán cómo se ejecutará el entrenamiento. Se procede entonces a construir los DataLoaders para los conjuntos de entrenamiento y validación, así como a inicializar los modelos y la función de pérdida. Se configura también el optimizador y un *scheduler* para ajustar dinámicamente la tasa de aprendizaje durante el entrenamiento.

El entrenamiento se ejecuta en un ciclo que abarca el número de épocas definido. Durante cada época, el modelo se pone en modo de entrenamiento y se llama a *train_epoch* para realizar el entrenamiento sobre los datos. Una vez completada la época de entrenamiento, se evalúa el modelo en el conjunto de validación, desactivando el cálculo de gradientes para reducir el consumo de memoria y asegurar que no se realicen modificaciones en los pesos del modelo durante la evaluación. Las pérdidas promedio del entrenamiento y la validación se registran en W&B, lo que permite un seguimiento detallado del progreso y facilita la toma de decisiones para ajustes en tiempo real. El *scheduler* ajusta la tasa de aprendizaje según la estrategia definida, ayudando al modelo a converger más eficazmente. Al finalizar todas las épocas y configuraciones del *sweep*, el agente de W&B

concluye el proceso, habiendo registrado todos los resultados y métricas relevantes, lo que proporciona una visión clara de las configuraciones más efectivas para el modelo.

Lancé el proceso de búsqueda de hiperparámetros con 20 ejecuciones, lo que implicó realizar 20 entrenamientos distintos del modelo, cada uno con diferentes combinaciones de parámetros. Este proceso, conocido como *sweep*, es crucial para encontrar la configuración óptima de hiperparámetros como el tamaño del lote, la tasa de aprendizaje y el optimizador. Aunque cada ejecución es un "mini entrenamiento", el objetivo es identificar cuál ofrece los mejores resultados en términos de pérdida y precisión.

En mi caso, el *sweep* duró dos horas. Este tiempo invertido es valioso, ya que al final se obtiene una configuración optimizada de hiperparámetros, mejorando así el rendimiento del modelo y ahorrando tiempo en futuros entrenamientos. A continuación, en la Figura 4.23., se puede observar la pérdida obtenida con cada combinación de hiperparámetros.

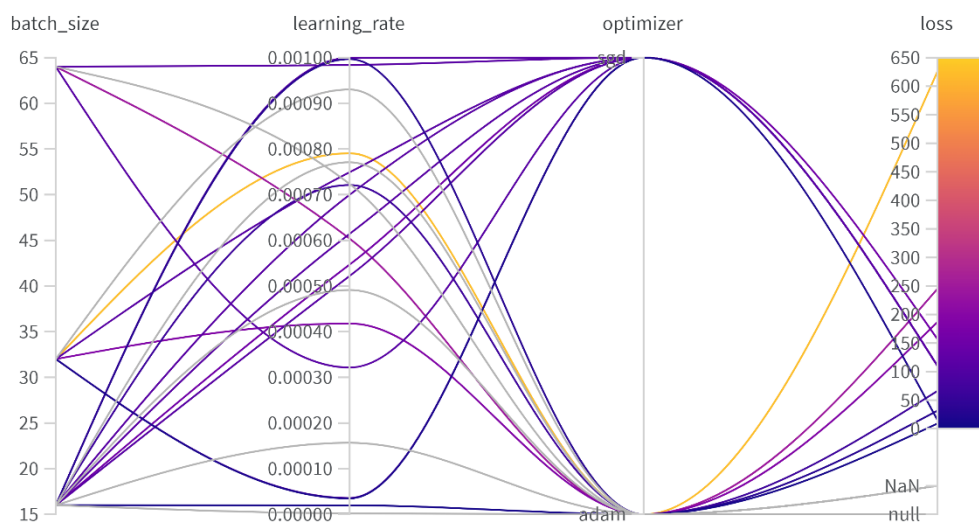


Figura 4.23. Combinación de hiperparámetros para minimizar el *loss*

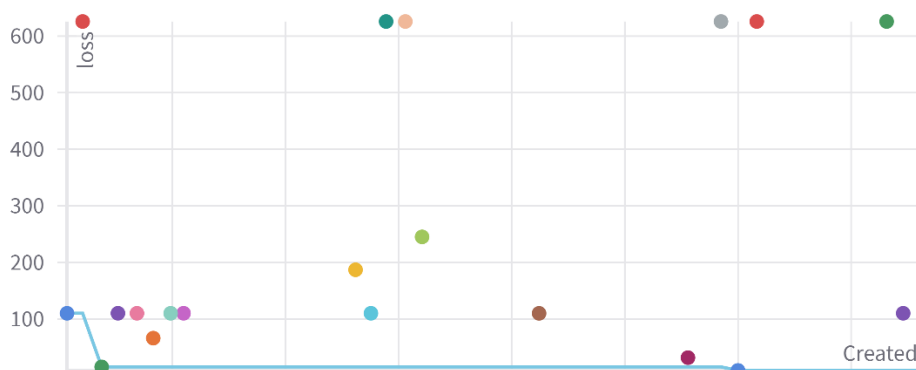


Figura 4.24. Eje cronológico y su resultado en cuanto a pérdida

Uno de los gráficos que más valor nos aporta a la hora de elegir la combinación de hiperparámetros para el entrenamiento final es el mostrado en la Figura 4.25. Este

proporciona información valiosa sobre la importancia y correlación de los diferentes hiperparámetros con respecto a la pérdida (loss) del modelo.

1. Tasa de aprendizaje (learning_rate):

- **Importancia:** Este parámetro tiene una alta importancia, lo que indica que es un factor crítico en la predicción de la pérdida del modelo. Una tasa de aprendizaje adecuada es crucial para el rendimiento general del modelo, ya que controla la velocidad con la que el modelo actualiza sus pesos durante el entrenamiento.
- **Correlación:** La correlación positiva sugiere que, en general, a medida que aumenta la tasa de aprendizaje dentro de los rangos explorados, la pérdida también tiende a aumentar. Esto es un indicativo de que una tasa de aprendizaje más baja podría ser más efectiva para minimizar la pérdida.

2. Tamaño de lote (batch_size):

- **Importancia:** El tamaño del lote también muestra una importancia significativa, lo que refleja su influencia en la estabilidad y eficiencia del entrenamiento.
- **Correlación:** La correlación positiva indica que, en los rangos explorados, tamaños de lote más grandes podrían estar asociados con un incremento en la pérdida, lo que sugiere que tamaños de lote más pequeños o medianos podrían ser más eficaces para mejorar el rendimiento del modelo.

3. Optimizadores (optimizer:sgd y adam):

- **Importancia:** Los optimizadores muestran menos importancia en comparación con la tasa de aprendizaje y el tamaño del lote.
- **Correlación:** La correlación negativa para SGD y positiva para Adam implica que, dentro de los rangos de parámetros probados, el uso de SGD tiende a asociarse con una reducción en la pérdida, mientras que Adam puede no ser tan efectivo en comparación con SGD en este contexto específico. Sin embargo, la magnitud de esta correlación no es extrema, lo que sugiere que ambos optimizadores pueden ser viables dependiendo de la configuración de otros hiperparámetros.

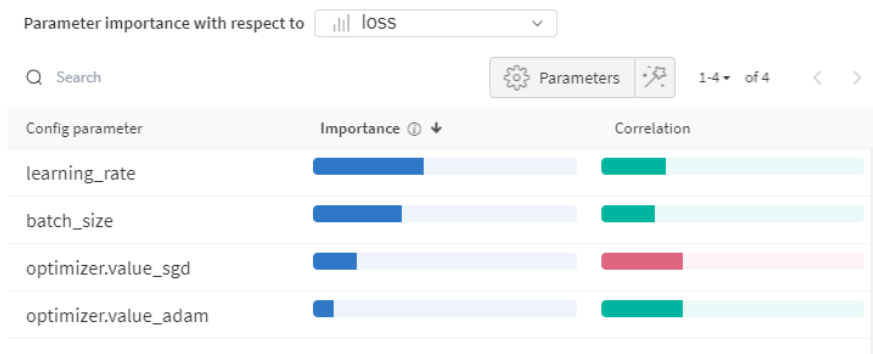


Figura 4.25. Importancia y correlación de hiperparámetros

Dado estos resultados, la elección final de utilizar un tamaño de lote de 32, el optimizador Adam, y una tasa de aprendizaje de 0.00001 es sólida y bien fundamentada:

- **Tasa de Aprendizaje (0.00001):** Optar por una tasa de aprendizaje baja es una decisión prudente, especialmente considerando la correlación positiva observada con la pérdida. Esto permite que el modelo ajuste sus parámetros de manera más gradual y precisa, reduciendo el riesgo de saltarse mínimos locales y facilitando una convergencia más estable.
- **Tamaño del Lote (32):** Un tamaño de lote de 32 es un buen compromiso entre la precisión de las estimaciones del gradiente y la estabilidad del entrenamiento. Este tamaño proporciona un equilibrio óptimo entre la eficiencia computacional y la capacidad del modelo para generalizar bien, sin incurrir en la inestabilidad que podría surgir con lotes demasiado pequeños o la lentitud de lotes demasiado grandes.
- **Optimizador (Adam):** Aunque SGD mostró una correlación negativa con la pérdida, Adam sigue siendo una excelente opción debido a su capacidad para ajustar dinámicamente las tasas de aprendizaje de manera efectiva. Adam es especialmente útil en configuraciones donde las tasas de aprendizaje son bajas, ya que puede manejar ajustes finos de los parámetros, lo que es crucial para optimizar modelos complejos.

4.4.3.2 Entrenamiento del modelo

El modelo se va a entrenar utilizando un tamaño de lote de 32, el optimizador Adam, y una tasa de aprendizaje de 0.00001. Una vez que se inicia el proceso de entrenamiento, el primer paso es determinar si se va a continuar desde un *checkpoint* previo o si se empieza desde cero. Si se decide continuar desde un *checkpoint*, se cargan el último *checkpoint* guardado y el mejor modelo basado en la menor pérdida alcanzada hasta el momento. Esto incluye cargar los pesos del modelo, el estado del optimizador, la pérdida registrada y la época en la que se interrumpió el entrenamiento. Para esto tenemos la función de *load_checkpoint*.

En cuanto al entrenamiento se usan las mismas líneas que en la notebook anterior visibles en la Figura 4.22., esto se debe a que cuando hicimos la búsqueda de hiperparámetros mediante sweeps, realmente están ocurriendo “mini entrenamientos”.

Durante el entrenamiento, por cada época se entrena el modelo utilizando los datos disponibles en los *loaders* de entrenamiento. En cada iteración, el modelo realiza una predicción, se calcula la pérdida utilizando la función de pérdida EPELoss, y luego se retropropaga esta pérdida para ajustar los pesos del modelo. Al final de cada época, se realiza una validación del modelo en un conjunto de datos independiente para evaluar su rendimiento general.

El proceso de guardado de *checkpoints* es clave durante el entrenamiento. Se manejan dos tipos de *checkpoints*: uno que guarda el estado después de cada número predefinido de épocas, y otro que guarda el mejor modelo alcanzado hasta el momento en función de la pérdida mínima. Esto asegura que siempre se tenga una versión reciente del modelo y que no se pierda progreso en caso de interrupciones, además de conservar la mejor versión del modelo entrenado. Para realizar estos guardados tenemos las funciones

save_checkpoint y *save_best_checkpoint*, cada una de ellas funcionan similar, la primera de ellas guardando el ultimo *checkpoint* con un *path* del tipo “last_checkpoint_epoch_{epoch}.pth” y borrando el anterior *checkpoint* donde se guarda la época, *model_1_state_dict*, *model_2_state_dict*, *optimizer_state_dict* y el *loss*. La función *save_best_checkpoint*, guarda los mismos valores en el *path* “best_checkpoint.pth”.

```
# Save checkpoint every checkpoint_interval epochs
if (epoch + 1) % checkpoint_interval == 0:
    save_checkpoint(epoch + 1, model_1, model_2, optimizer, avg_train_loss, checkpoint_dir)
# Save the best checkpoint
if avg_train_loss < best_loss:
    best_loss = avg_train_loss
    save_best_checkpoint(epoch, model_1, model_2, optimizer, avg_train_loss, checkpoint_dir)
```

Figura 4.26. Guardado del ultimo y mejor *checkpoint*

Al finalizar el entrenamiento, se selecciona como modelo final el mejor *checkpoint* guardado, el cual representa la mejor capacidad del modelo para generalizar los datos, basándose en la pérdida más baja obtenida durante el entrenamiento. Este modelo final es el que se guarda para futuras inferencias o evaluaciones.

4.4.3.3 Resumen resultados

El entrenamiento se llevó a cabo en un sistema con una GPU NVIDIA GeForce RTX 4080. La sesión de entrenamiento tuvo una duración total de 3 horas, 5 minutos y 21 segundos, durante los cuales se ejecutaron 50 épocas completas. El conjunto de datos utilizado consistió en 22000 imágenes con un tamaño de 256x256 píxeles, procesadas en lotes de 32 imágenes por iteración.

En términos de configuración de los hiperparámetros, el modelo se entrenó con una tasa de aprendizaje de 0.00001, un tamaño de lote de 32 y un total de 625 lotes en el conjunto de entrenamiento (train loader) y 62 lotes en el conjunto de validación (val loader). Durante todo el entrenamiento, se completaron 31,245 pasos, lo que da una idea de la magnitud del proceso.

Al final de las 50 épocas, las métricas reflejan un progreso significativo en la optimización del modelo. La pérdida promedio en el conjunto de entrenamiento (epoch/avg_train_loss) se redujo a 2.156, lo que indica que el modelo fue capaz de capturar y aprender patrones relevantes de los datos de manera consistente y eficiente.

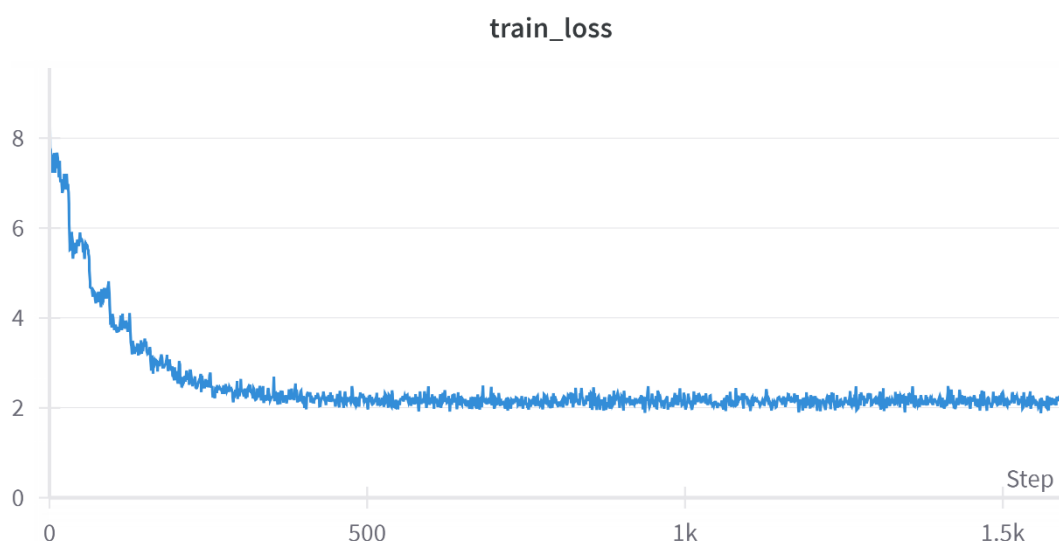


Figura 4.27. Descenso del *train_loss* durante el entrenamiento

Además, todo el proceso de entrenamiento fue cuidadosamente monitorizado y registrado en tiempo real mediante la plataforma Weight & Biases. Esto permitió realizar un seguimiento detallado de las métricas clave, como las pérdidas por época, y almacenar *logs* completos de cada etapa del entrenamiento. Este registro minucioso facilita no solo la revisión y el análisis de los resultados, sino también la posibilidad de comparar diferentes experimentos y ajustes, mejorando así la toma de decisiones en futuras iteraciones del modelo.

```
1553 Iter 540 Loss 2.261805349588394
1554 Iter 560 Loss 2.079525262117386
1555 Iter 580 Loss 2.2752385199069978
1556 Iter 600 Loss 2.075583791732788
1557 Iter 620 Loss 2.1250286281108854
1558 Average Epoch Loss 2.164532021331787
1559 EPOCH 44, LOSS train 2.164532021331787 LOSS val 6.216970920562744
1560 Last checkpoint saved at C:/TFG/joel/ImageEnhancementTFG/models/GeoProj/checkpoints/last_checkpoint_epoch_45.pth
1561 Epoch 45
1562 Iter 20 Loss 2.2617886483669283
1563 Iter 40 Loss 2.2020671129226685
```

Figura 4.28. Pequeña muestra de *logs* en Weight&Bias

4.4.4 Inferencia en notebook

Se ha desarrollado una nueva notebook de inferencia que facilita el proceso de evaluación de los distintos modelos generados durante el entrenamiento. En esta notebook, se cargan los pesos de los modelos entrenados utilizando la función *load_weights*, que asegura que los modelos se carguen correctamente en la GPU si está disponible, o en la CPU en su defecto.

Para procesar las imágenes antes de realizar la inferencia, primero es necesario recortarlas para que tengan una forma cuadrada, preservando la dimensión más grande. Luego, se redimensionan a 256x256 píxeles y se normalizan mediante una serie de transformaciones definidas en la función *transform_image*. Esta función convierte la imagen a un tensor compatible con PyTorch, listo para ser procesado por los modelos.

Una vez transformada la imagen, se pasa por los modelos `model_1` y `model_2` para generar una salida de flujo óptico, que luego se utiliza para rectificar la distorsión de la imagen original a través de la función `rectify_image`. Este flujo de trabajo ha permitido probar y comparar los diferentes modelos de manera eficiente, asegurando que cada versión se ajuste a los nuevos datos de la mejor manera posible. A continuación, dejo un ejemplo del modelo final y su funcionamiento:



Figura 4.29. Ejemplo de aplicación del modelo entrenado

4.4.5 Implementación del modelo

La implementación del modelo de corrección de distorsión de lente se ha realizado de manera eficiente, en gran parte gracias a la estructura proporcionada por la clase `LensDistortion`, que es una subclase de la clase abstracta `Model`. Esta arquitectura facilita la integración de nuevos modelos en la aplicación, manteniendo el código limpio y modular.

Dentro de la clase `LensDistortion`, se ha incorporado una mejora significativa en la salida de las imágenes corregidas mediante el uso del archivo `utils.py`. Este archivo implementa una función de súper resolución utilizando ESRGAN (Enhanced Super Resolution GAN) [17] [18], que es crucial para mejorar la resolución original de 256x256 píxeles producida por el modelo de distorsión de lente.

El proceso comienza con la carga del modelo ESRGAN mediante la función `load_esrgan_model`. Posteriormente, la función `super_resolution` se encarga de procesar la imagen corregida para aumentar su resolución, asegurando que el resultado final sea visualmente detallado y de alta calidad. Además, el pre procesamiento de la imagen, que incluye recortes y ajustes de dimensiones, garantiza que la súper resolución se aplique de manera efectiva.

Gracias a estas mejoras, las imágenes resultantes no solo están libres de distorsión, sino que también presentan una calidad visual superior, lo que es fundamental para aplicaciones donde la precisión y la claridad son esenciales.

```
def super_resolution(input_image: np.ndarray) -> np.ndarray:
    """Perform super resolution on the input image using ESRGAN..."""

    # Load ESRGAN model
    esrgan_model = load_esrgan_model(config['models']['super_resolution_model'])

    # Perform super resolution
    preprocessed_image = preprocessing(input_image)
    enhanced_image_tf = esrgan_model(preprocessed_image)
    enhanced_image_np = tf.squeeze(enhanced_image_tf).numpy().astype(np.uint8)
    logging.info(f"Super resolution successful")

    return enhanced_image_np
```

Figura 4.30. Aplicación del modelo de súper resolución ESRGAN

4.5 Fase 5: Documentación del código

4.5.1 Documentación

La documentación es un componente fundamental en el desarrollo de software, en este proyecto, se ha optado por utilizar el formato de *docstring* de NumPy/SciPy, un estándar ampliamente reconocido en la comunidad científica y de desarrollo. Este formato proporciona una estructura clara y detallada para describir el propósito, los parámetros y los valores de retorno de cada función o método.

El formato NumPy/SciPy organiza la información de manera que cualquier desarrollador, presente o futuro, pueda entender rápidamente cómo utilizar una función y qué esperar de ella. Las secciones comunes como "Parameters", "Returns" y "Notes" se incluyen para garantizar que todos los aspectos relevantes de la función estén documentados. Esta metodología no solo facilita la comprensión del código, sino que también ayuda a reducir la cantidad de errores y malentendidos al hacer que las expectativas sobre la entrada y salida de datos sean explícitas. La documentación exhaustiva asegura que el código sea fácilmente accesible para otros miembros del equipo y para los desarrolladores que puedan trabajar en el proyecto en el futuro. En la Figura 4.31. muestro un ejemplo de documentación de una función y en la Figura 4.32.

```
def apply_transformations(input_images: List[str], options: List[str], model_manager: ModelManager, sky_image: Optional[np.ndarray] = None) -> List[np.ndarray]:
    """
    Apply transformations to input images based on selected options.

    Parameters:
        input_images (List[str]): List of input image paths.
        options (List[str]): List of selected options.
        model_manager (ModelManager): Model manager instance.
        sky_image (Optional[np.ndarray]): Sky image for replacement. Defaults to None.

    Returns:
        List[np.ndarray]: List of enhanced images as NumPy arrays.
    """
```

Figura 4.31. Docstring de la función *apply_transformations*

```
class Llama:
    """
    A class to interact with the llama API for generating responses based on text and image inputs.

    Attributes
    -----
    config : dict
        Configuration loaded from 'data/config.ini'.
    llama_api : LlamaAPI
        Instance of the LlamaAPI.
    prompts : dict
        Dictionary containing the prompts.

    Methods
    -----
    generate(message: Optional[str], image: Optional[bool]) -> LlamaResponse:
        Generate a response based on the message and image provided.
    handle_image_and_text(message: str) -> LlamaResponse:
        Handle the case where both image and text are provided.
    handle_text(message: str) -> LlamaResponse:
        Handle the case where only text is provided.
    handle_image_only() -> LlamaResponse:
        Handle the case where only an image is provided.
    fetch_response_content(api_json: dict) -> LlamaResponse:
        Fetch the response content from the API.
    process_image_and_text_response(response_dict: dict, message: str) -> LlamaResponse:
        Process the response for image and text input.
    response_to_no_tasks(message: str) -> LlamaResponse:
        Generate a response when no tasks are detected.
    handle_error(error: Exception) -> LlamaResponse:
        Handle any errors that occur during processing.
    """
```

Figura 4.32. Docstring de la clase Llama

4.5.2 Tipado

El tipado de funciones es otro pilar clave que se ha implementado con rigor en este proyecto. A pesar de que Python es un lenguaje dinámico, el uso de anotaciones de tipo (type hints) agrega una capa adicional de claridad y seguridad al código. Especificar los tipos de datos esperados para los parámetros de entrada y los valores de retorno de las funciones no solo mejora la legibilidad del código, sino que también proporciona garantías adicionales sobre su correcto funcionamiento.

El tipado explícito es especialmente útil cuando se trabaja en equipos, ya que establece un contrato claro entre el desarrollador que escribe una función y los que la utilizan. Además, facilita el uso de herramientas de análisis estático como mypy, que pueden detectar inconsistencias de tipo antes de que el código se ejecute. Esto no solo previene errores comunes, sino que también mejora la robustez del sistema en su conjunto. En un entorno donde la precisión y la estabilidad son críticas, como en el procesamiento de imágenes y el aprendizaje profundo, el tipado cuidadoso es una práctica que eleva significativamente la calidad del código. En la Figura 4.31. se puede ver el ejemplo con la clase *apply_transformations* tanto de los parámetros de entrada como el valor de retorno.

4.5.3 Logging

El uso de *logging* en el código en lugar de simples declaraciones *print* representa una práctica avanzada que añade un nivel de profesionalismo y control al desarrollo del software. Mientras que *print* es útil para depuración básica, *logging* ofrece una gama

mucho más amplia de funcionalidades, incluyendo la capacidad de registrar eventos en diferentes niveles de gravedad (DEBUG, INFO, WARNING, ERROR, CRITICAL), redirigir la salida a archivos, y gestionar el formato de los mensajes de manera centralizada.

En este proyecto, el *logging* se utiliza para monitorear el flujo de ejecución, diagnosticar problemas y documentar eventos importantes. Al implementar un sistema de *logging* robusto, el código se vuelve más fácil de mantener y depurar, especialmente en situaciones donde el comportamiento erróneo podría no ser inmediatamente obvio. Por ejemplo, en tareas de procesamiento de imágenes, donde el rendimiento y la precisión son críticos, el *logging* permite a los desarrolladores identificar rápidamente dónde y por qué ocurre un error, facilitando una respuesta rápida y eficaz.

En resumen, la combinación de una documentación detallada, un tipado explícito y un sistema de *logging* eficaz no solo mejora la calidad y la mantenibilidad del código, sino que también establece un estándar elevado de desarrollo que es crucial para el éxito a largo plazo del proyecto.

```
INFO:root:Only text provided. Processing text input...
INFO:root:Calling llama API
INFO:root:LLM Response: LlamaResponse(tasks=None, new_background_image=None, response_text='¡Hola! ¿En qué puedo ayudarte con una imagen hoy?', error_message=None)
INFO:root:Both image and text provided. Processing image and text input...
INFO:root:Calling llama API
INFO:root:LLM Response: LlamaResponse(tasks=['cambiando el cielo'], new_background_image='sunset wallpaper', response_text=None, error_message=None)
INFO:root:Loading Sky Replacement model
INFO:root:Using device cuda
C:\Users\JoelVP\anaconda3\envs\ImageEnhancementTF6\lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0
warnings.warn(
C:\Users\JoelVP\anaconda3\envs\ImageEnhancementTF6\lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other than a weight enum or 'None' for
warnings.warn(msg)
INFO:root:Sky Replacement model loaded
INFO:root:Applying Sky Replacement
INFO:root:CUDA cache cleared
INFO:root:Initialize skybox...
INFO:root:Sky Replacement applied!
INFO:root:Images enhanced correctly!
```

Figura 4.33. Ejemplo de la terminal en una ejecución del programa

Capítulo 5. Conclusiones y propuesta de trabajo futuro

5.1 Conclusiones

El desarrollo de esta herramienta representa un hito significativo en el campo del procesamiento de imágenes asistido por inteligencia artificial. La versatilidad de la solución creada la posiciona como una herramienta valiosa tanto para usuarios particulares como para empresas. A nivel individual, la herramienta permite a cualquier persona mejorar la calidad de sus fotografías utilizando modelos avanzados de inteligencia artificial. La facilidad de uso y la potencia de los modelos hacen posible que cualquier usuario obtenga resultados de alta calidad con un esfuerzo mínimo.

Desde una perspectiva empresarial, la herramienta ofrece la capacidad de realizar un filtrado previo eficiente de grandes volúmenes de imágenes. Esta funcionalidad es especialmente útil en sectores como la publicidad, la fotografía profesional, y el marketing digital, donde la calidad visual es crucial. Al automatizar el proceso de mejora de imágenes, las empresas pueden optimizar sus flujos de trabajo, ahorrar tiempo y reducir costos, permitiendo a los equipos creativos centrarse en tareas de mayor valor añadido. Esta doble aplicabilidad, tanto en el ámbito personal como empresarial, subraya la relevancia y utilidad de la herramienta en el contexto actual.

Además, uno de los aspectos más destacables del proyecto es el enfoque profesional que se ha mantenido en todas las fases de desarrollo. La integración de herramientas de control de versiones como Git asegura que el desarrollo del software sea manejado de manera eficiente y colaborativa, permitiendo un seguimiento detallado de los cambios y facilitando la colaboración entre diferentes desarrolladores, si fuera un proyecto real. El compromiso con la documentación exhaustiva y clara, el uso de tipado explícito en Python, y la implementación de un sistema de *logging* robusto reflejan una atención meticulosa al detalle que no solo mejora la calidad del código, sino que también garantiza su mantenibilidad a largo plazo.

El proceso de desarrollo también se ha distinguido por su enfoque end-to-end, abarcando desde la construcción del backend y frontend hasta la aplicación de técnicas avanzadas de data science. La capacidad para entrenar un modelo con un dataset propio demuestra un nivel avanzado de competencia técnica y asegura que la herramienta esté personalizada y optimizada para las necesidades específicas del proyecto. Además, la abstracción de clases y funciones se ha implementado cuidadosamente para facilitar la futura expansión del sistema, permitiendo la incorporación de nuevos modelos de manera ágil y eficiente.

Finalmente, la integración de un chatbot basado en modelos de lenguaje grande (LLM) representa un elemento de innovación que eleva la herramienta a un nivel aún más avanzado. Este componente no solo mejora la experiencia del usuario al permitir interacciones más naturales y fluidas, sino que también amplía las capacidades del sistema al permitir una mayor automatización y personalización en el procesamiento de imágenes.

5.2 Propuesta de trabajo futuro

En cuanto a las futuras direcciones del proyecto, existen varias áreas clave que podrían mejorar significativamente la funcionalidad, escalabilidad y eficacia de la herramienta desarrollada.

Una de las principales áreas de mejora es la migración de la infraestructura a la nube. Actualmente, la herramienta podría beneficiarse enormemente de estar alojada en un entorno de servidor en la nube, como AWS, Azure o Google Cloud. Este cambio no solo garantizaría que la página web esté disponible de manera constante y fiable, sino que también permitiría una escalabilidad prácticamente ilimitada en función de la demanda de los usuarios. Con la infraestructura en la nube, los modelos de IA podrían estar desplegados y optimizados en herramientas como MLflow para un acceso rápido y eficiente, mejorando la experiencia del usuario final y garantizando que la herramienta esté siempre lista para operar sin interrupciones.

Otra área de mejora significativa es el entrenamiento del modelo de distorsión de lente con un conjunto de datos más amplio y en un entorno computacional más potente. Inicialmente, el entrenamiento se llevó a cabo utilizando una GPU limitada, lo que restringió tanto la cantidad de datos como la complejidad de los modelos que podían manejarse. Aunque se logró realizar un entrenamiento más adecuado utilizando un equipo de la ETSIT, el tiempo limitado en este entorno impidió un perfeccionamiento óptimo del modelo. Ampliar el conjunto de datos y utilizar un entorno de entrenamiento más robusto, quizás aprovechando las capacidades de la nube, permitiría obtener un modelo más preciso y generalizable, capaz de corregir distorsiones de lente en una gama más amplia de imágenes.

Además, se plantea la necesidad de reentrenar todos los modelos con conjuntos de datos propios. El uso de *datasets* personalizados no solo mejoraría la precisión y relevancia de los modelos para las necesidades específicas del proyecto, sino que también permitiría un mayor control sobre la calidad y diversidad de los datos, asegurando que los modelos se adapten mejor a los casos de uso previstos.

En relación con los modelos de lenguaje, un paso significativo sería adoptar una LLM (Large Language Model) más potente para el chatbot integrado en la herramienta. El uso de un modelo más avanzado permitiría mejorar la interacción con los usuarios, haciendo que el chatbot sea más competente en comprender y responder a una gama más amplia de consultas de manera más natural y precisa. Esto no solo mejoraría la experiencia del usuario, sino que también podría habilitar nuevas funcionalidades basadas en el procesamiento avanzado del lenguaje.

Una propuesta adicional sería la implementación de una arquitectura de micro servicios. Este enfoque permitiría dividir la aplicación en componentes más pequeños e independientes, facilitando su mantenimiento, escalabilidad y desarrollo continuo. La arquitectura de micro servicios también podría permitir una mayor flexibilidad al integrar nuevas funciones o mejorar las existentes, ya que cada servicio puede ser desarrollado, desplegado y escalado de manera independiente.



Finalmente, sería beneficioso explorar la optimización del modelo para dispositivos móviles. Aunque la herramienta está orientada principalmente al uso en plataformas web, permitir que los usuarios procesen imágenes directamente desde sus dispositivos móviles podría aumentar significativamente la accesibilidad y el alcance de la herramienta, especialmente en un mundo donde el uso de smartphones es omnipresente.

En conjunto, estas propuestas representan un conjunto coherente de mejoras que no solo incrementarían la robustez y eficacia de la herramienta, sino que también abrirían nuevas posibilidades de uso y expansión, consolidando su posición como una solución avanzada en el campo del procesamiento de imágenes mediante inteligencia artificial.

Bibliografía

- [1] Y. B. & G. H. Yann LeCun, “Deep Learning,” *Nature*, vol. 521, no. 7553, pp. 436-444, 2015.
- [2] D. -. A. Team, “Deci.ai,” [Online]. Available: <https://deci.ai/blog/sota-dnns-overview/>.
- [3] V. d. P. Joel, “GitHub,” 2024. [Online]. Available: <https://github.com/joelvp/ImageEnhancementTFG>.
- [4] A. Amin, “Medium,” 1 5 2023. [Online]. Available: <https://medium.com/@abhay.pixolo/naming-conventions-for-git-branches-a-cheatsheet-8549feca2534>.
- [5] V. d. P. Joel, “GitHub,” 2024. [Online]. Available: <https://github.com/joelvp/ImageEnhancementTFG/blob/main/README.md>.
- [6] L. a. C. X. a. Z. X. a. S. J. Chen, “Simple Baselines for Image Restoration,” *arXiv preprint arXiv:2204.04676*, 2022.
- [7] L. a. C. X. a. Z. X. a. S. J. Chen, “GitHub,” 2022. [Online]. Available: <https://github.com/megvii-research/NAFNet?tab=readme-ov-file>.
- [8] Y. a. W. R. a. Y. W. a. L. H. a. C. L.-P. a. K. A. C. Wang, “Low-Light Image Enhancement with Normalizing Flow,” *arXiv preprint arXiv:2109.05923*, 2021.
- [9] Y. a. W. R. a. Y. W. a. L. H. a. C. L.-P. a. K. A. C. Wang, “GitHub,” 2021. [Online]. Available: <https://github.com/wyf0912/LLFlow>.
- [10] M. S. B. Mahmoud Afif, “Deep White-Balance Editing,” *CVPR*, 2020.
- [11] M. S. B. Mahmoud Afif, “GitHub,” 2020. [Online]. Available: https://github.com/mahmoudnafifi/Deep_White_Balance?tab=readme-ov-file.
- [12] Z. Zou, “Castle in the Sky: Dynamic Sky Replacement and Harmonization in Videos,” *arXiv preprint arXiv:2010.11800*, 2020.
- [13] Z. Zou, “GitHub,” 2020. [Online]. Available: <https://github.com/jiupinjia/SkyAR?tab=readme-ov-file>.
- [14] X. a. Z. B. a. S. P. V. a. L. J. Li, “GitHub,” 2019. [Online]. Available: <https://github.com/xiaoyu258/GeoProj?tab=readme-ov-file>.
- [15] X. a. Z. B. a. S. P. V. a. L. J. Li, “Blind Geometric Distortion Correction on Images Through Deep Learning,” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4855--4864, 2019.
- [16] A. L. A. K. A. O. a. A. T. B. Zhou, “Places: A 10 million Image Database for Scene Recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.
- [17] X. W. a. L. X. a. C. D. a. Y. Shan, “Real-ESRGAN: Training Real-World Blind Super-Resolution with Pure Synthetic Data,” *International Conference on Computer Vision Workshops (ICCVW)*, 2021.
- [18] X. W. a. L. X. a. C. D. a. Y. Shan, “GitHub,” 2021. [Online]. Available: <https://github.com/xinntao/Real-ESRGAN?tab=readme-ov-file>.

ANEXOS

El código completo se encuentra en el siguiente enlace de GitHub: [3]

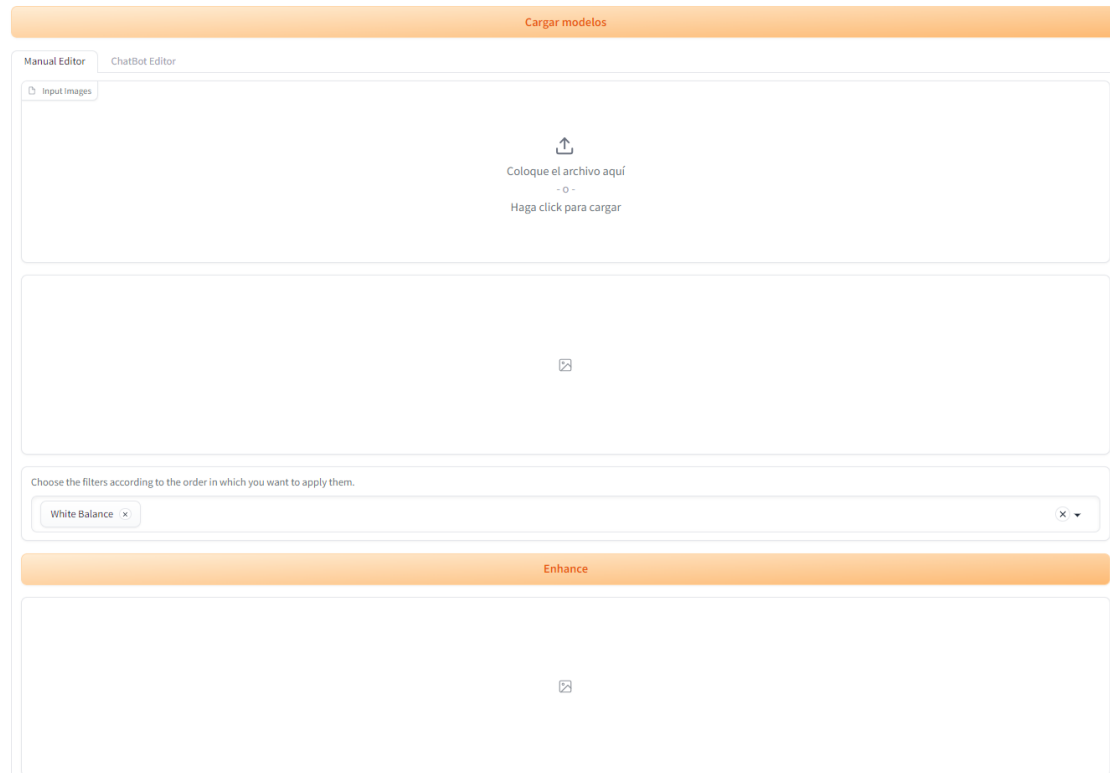


Figura 1. Pantalla inicial en editor manual

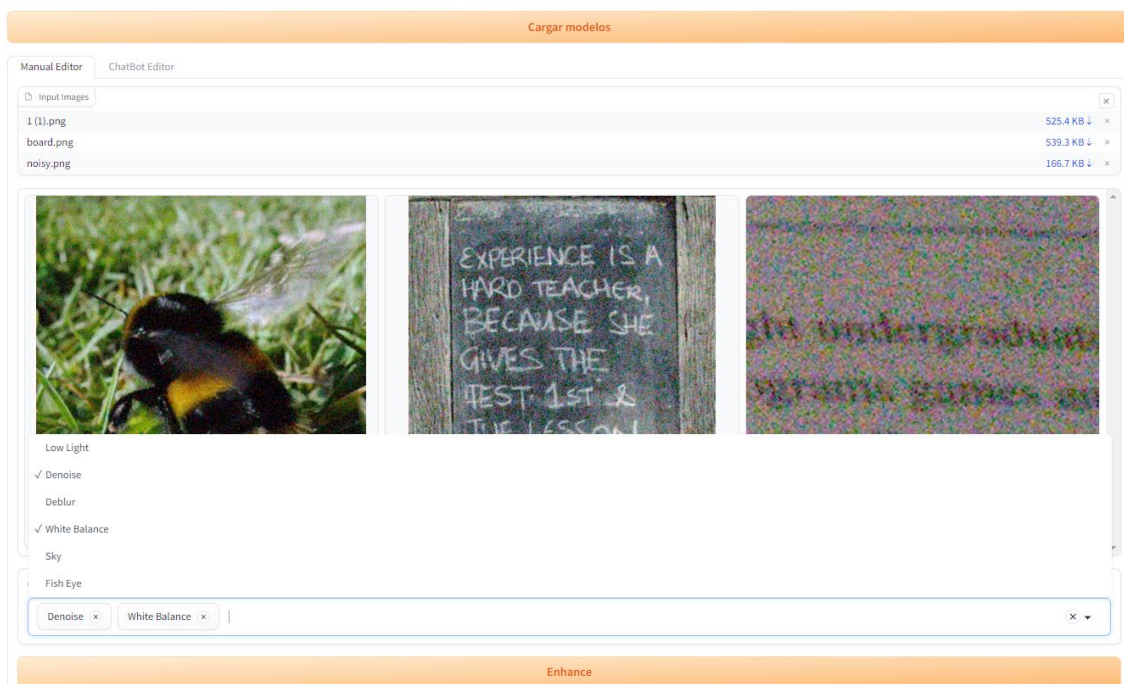


Figura 2. Editor manual con imágenes cargadas y menú de mejoras desplegado

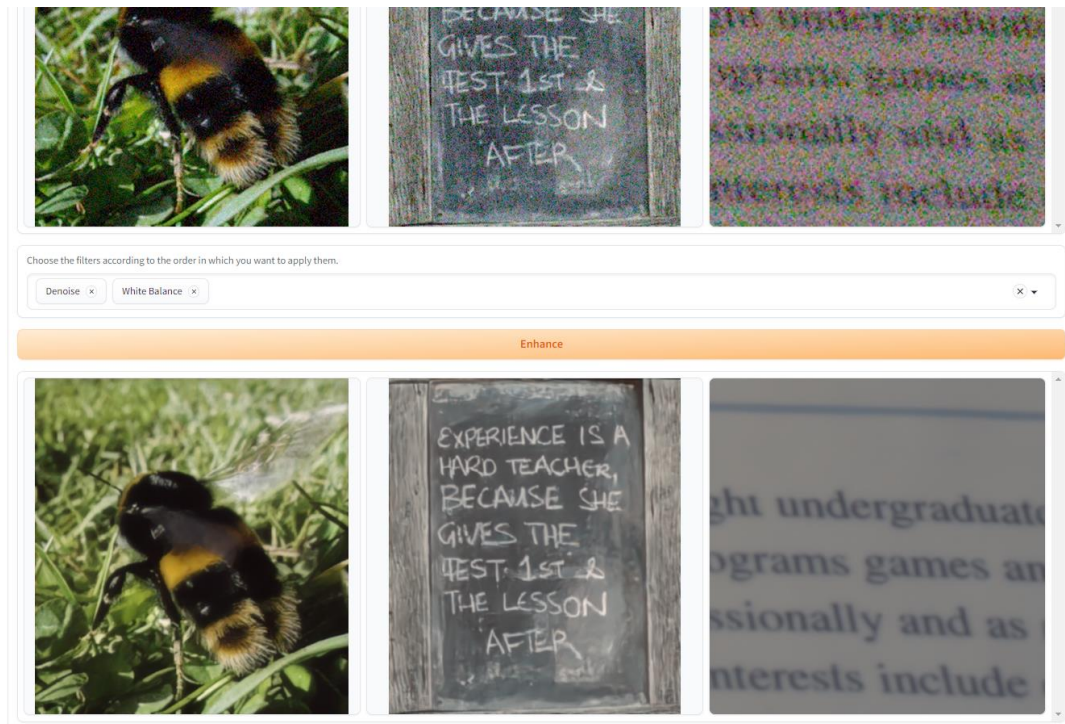


Figura 3. Editor manual con imágenes mejoradas

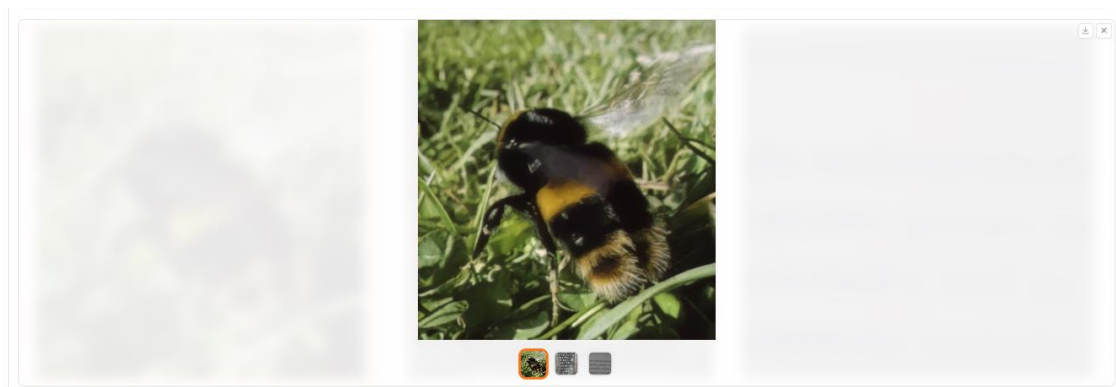


Figura 4. Imagen mejorada con opción de descarga en esquina superior derecha

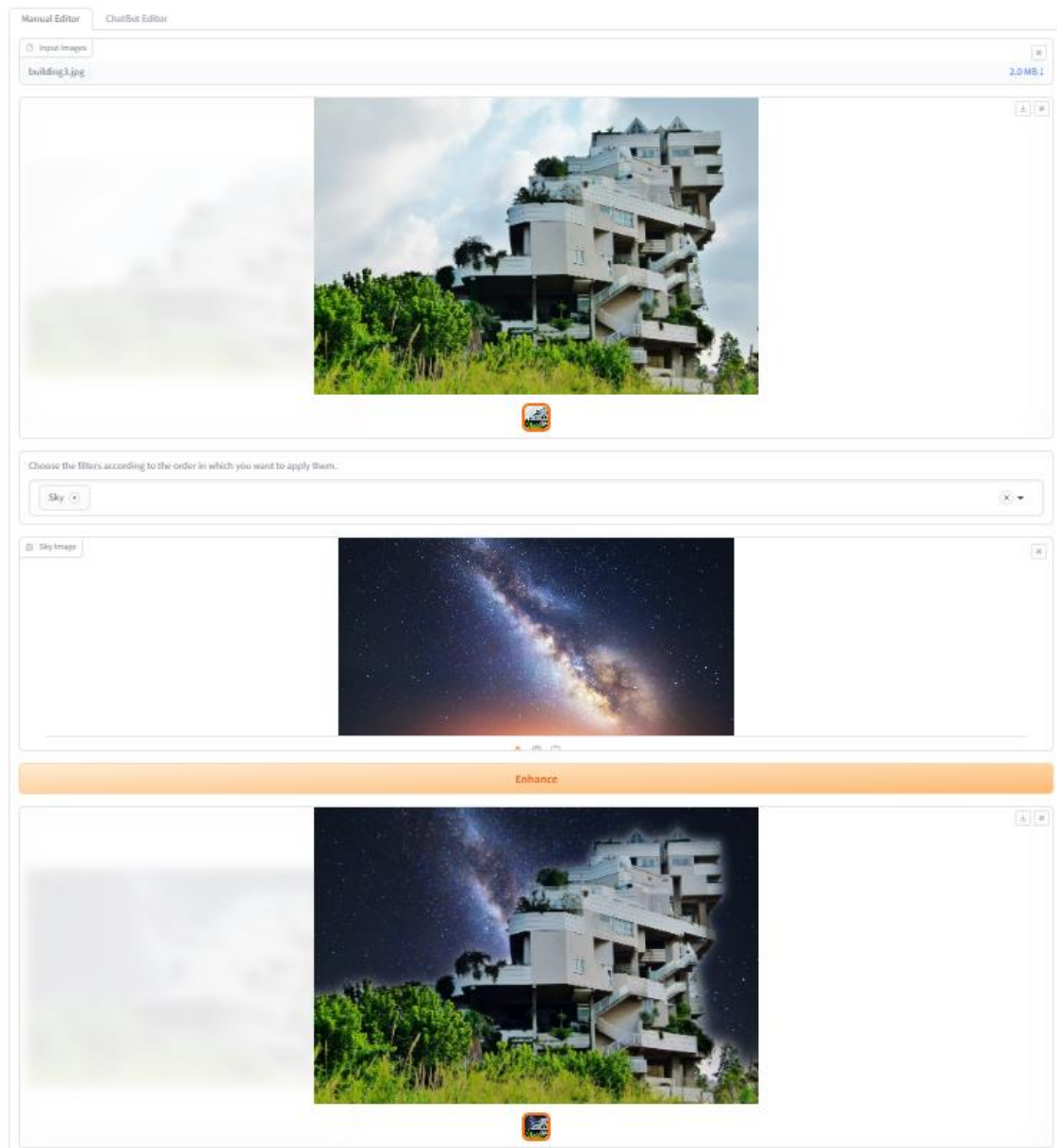


Figura 5. Editor manual con la opción de cambio de cielo y el fondo sugerido

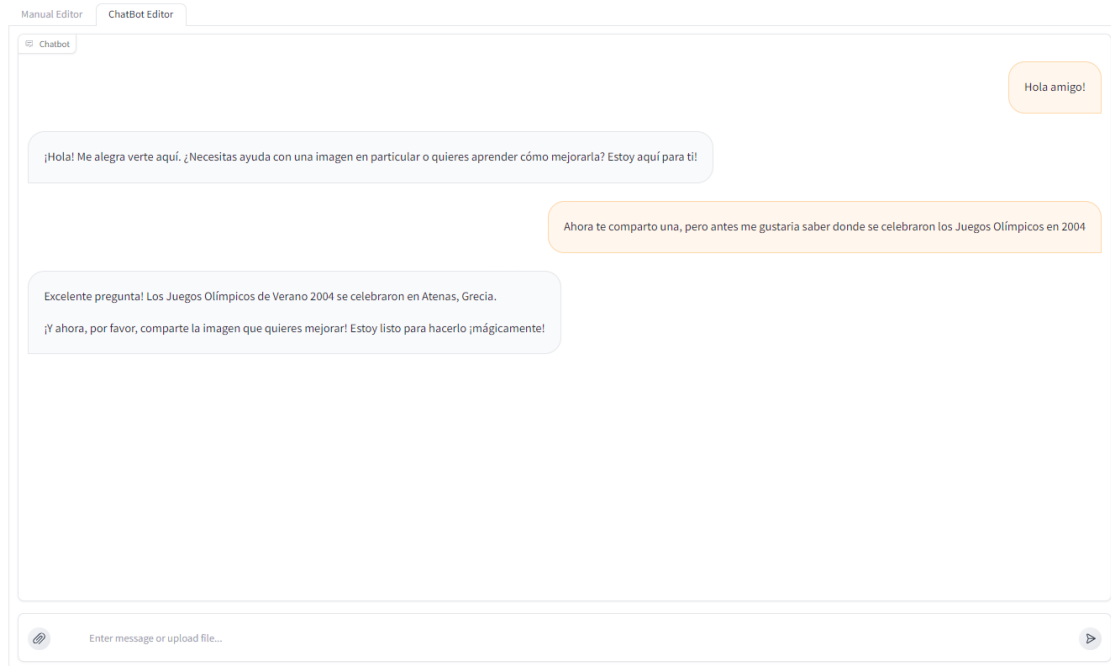


Figura 6. Conversación en lenguaje natural en el ChatBot

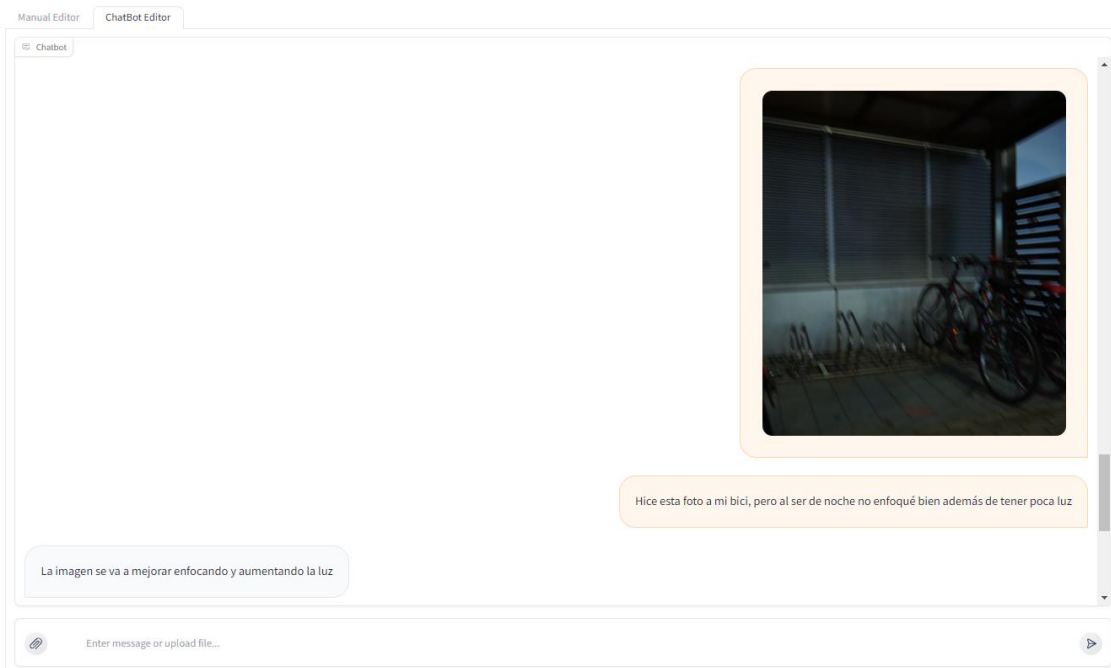


Figura 7. Imagen a mejorar con instrucciones en lenguaje natural

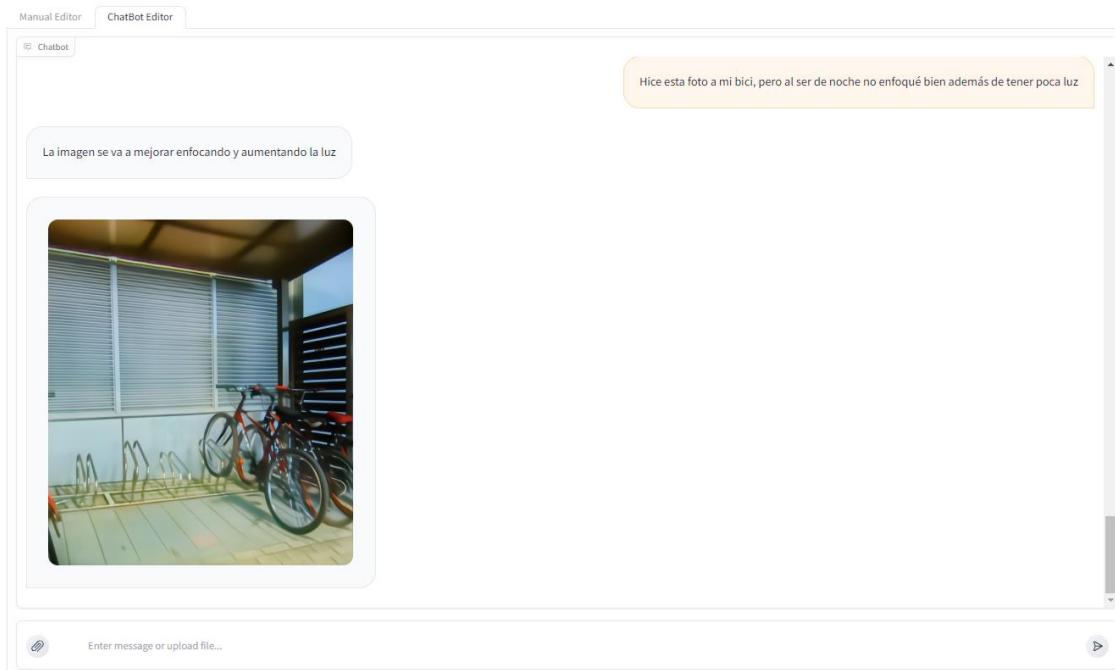


Figura 8. Respuesta de la LLM con texto explicativo e imagen mejorada

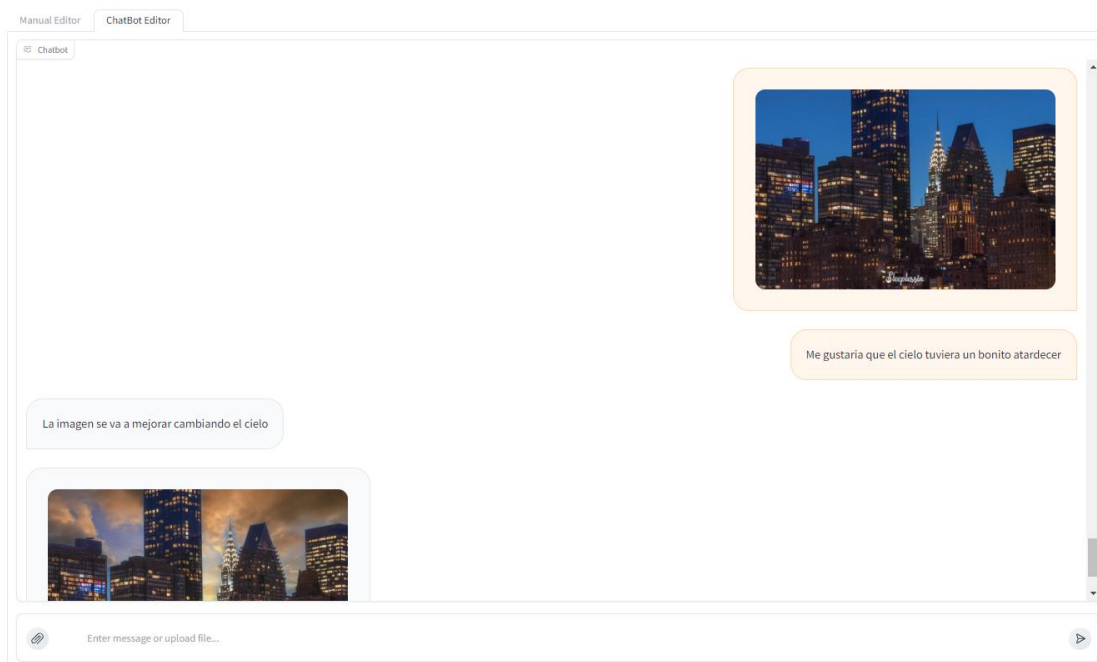


Figura 9. Petición de reemplazo de cielo con lenguaje natural

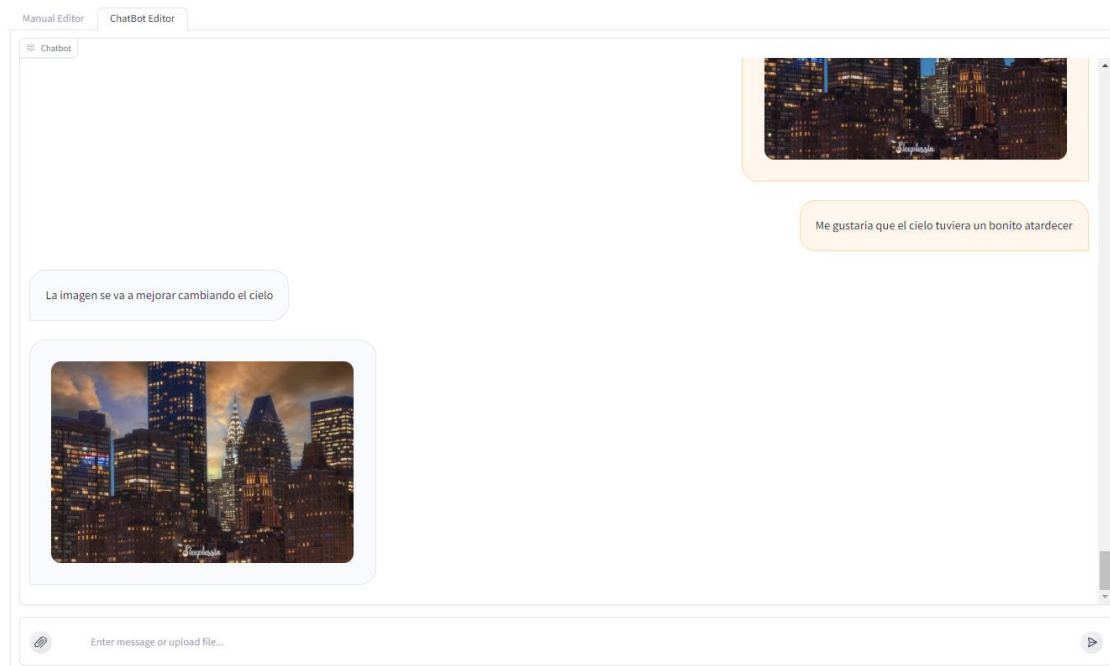


Figura 10. Imagen con el nuevo cielo en base a la petición

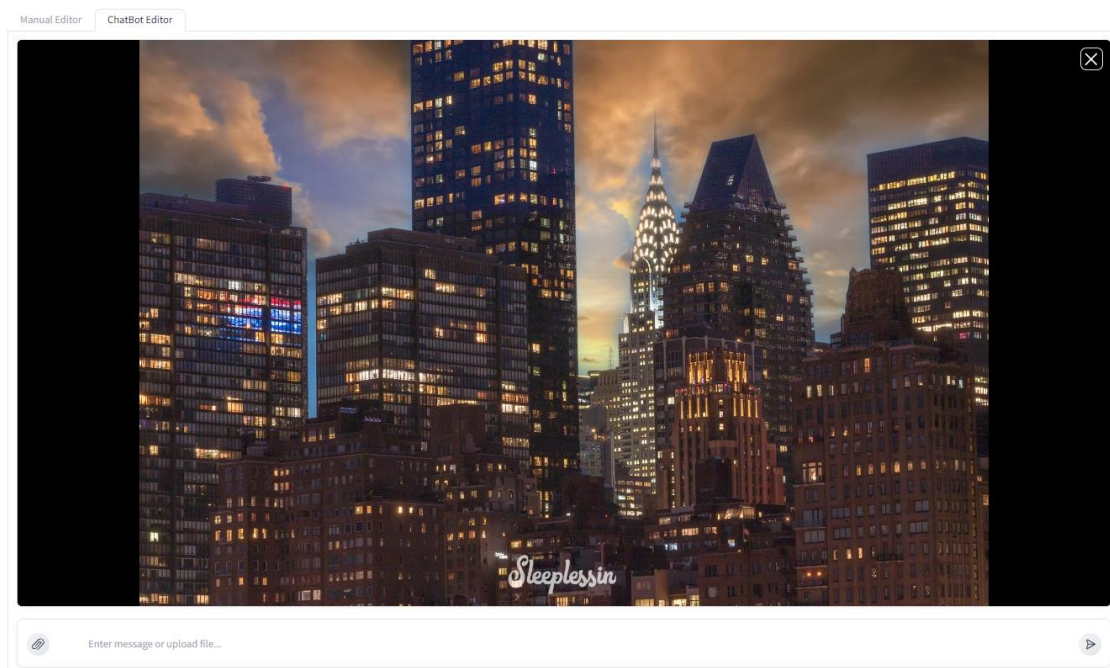


Figura 11. Posibilidad de ampliar y descargar imágenes

```
[api_keys]
llama = ***
google_search = ***
w&b = ***
[google_search]
cx = ***
[gradio]
background_image = models/sky_replace/background_img/galaxy.jpg
[llama]
prompts = data/prompts.json
[models]
lowlight_config = data/model_config/lowlight_config.yml
deblur_config = data/model_config/deblur_config.yml
denoise_config = data/model_config/denoise_config.yml
sky_replace_config = data/model_config/skyreplace_config.json
wb_model = models/white_balance/models
lens_distortion_model = models/lens_distortion/models
super_resolution_model = models/lens_distortion/models/esrgan
[lens_distortion]
checkpoints_dir = C:/Users/JoelVP/Desktop/UPV/ImageEnhancementTF6/imageenhancementtf6/models/lens_distortion/checkpoints
dataset_dir = E:/UPV/TF6/Places25k/Data_gen
```

Figura 12. Archivo de configuración *config.ini*

```
{
  "prompts": {
    "task_chooser": {
      "model": "mistral-8x22b-instruct",
      "temperature": 0,
      "messages": [],
      "functions": [
        {
          "name": "extraccion_informacion",
          "description": "Extrae la informacion relevante de la frase.",
          "parameters": {
            "type": "object",
            "properties": {
              "mejora_imagen": {
                "title": "mejora_imagen",
                "type": "boolean",
                "enum": ["True", "False"],
                "description": "True si la frase indica que la imagen necesita una mejora, False si no necesita la mejora o aunque la necesite no especifique como."
              },
              "tareas": {
                "title": "tareas",
                "type": "List(strings)",
                "description": "Las tareas necesarias para mejorar la imagen. Opciones : \\'quitando el ruido\\', \\'enfocando\\', \\'ajustando el balance de blancos\\', \\'aumentando la luz\\', \\'corrigiendo la distorsion de\\'"
              },
              "cambio_cielo": {
                "title": "cambio_cielo",
                "type": "boolean",
                "enum": ["True", "False"],
                "description": "True si la frase indica que se necesita cambiar el cielo/fondo de la imagen, False si no lo necesita"
              },
              "new_background_image": {
                "title": "new_background_image",
                "type": "string",
                "description": "El tema principal sobre el que debe ser el nuevo fondo de la imagen, incluye siempre 'wallpaper' y escríbelo en inglés, e.g. 'night sky with stars wallpaper'."
              }
            }
          }
        }
      ]
    },
    "stream": false,
    "function_call": {"name": "extraccion_informacion"}
  }
}
```

Figura 13. Prompt *task_chooser* para LLM

```
"plain_text": {
  "model": "llama3-8b",
  "messages": [
    {"role": "system", "content": "Eres un asistente que contesta brevemente a lo que te pregunten, pero recuerda SIEMPRE que tu estas aqui para ayudarles a mejorar imagenes"},
    {"role": "user", "content": "Hola!"},
    {"role": "assistant", "content": "Hola! Que deseas? Recuerda que estoy aqui para mejorar tus imagenes magicamente!"},
    {"role": "user", "content": "¿Cual es tu deporte favorito?"},
    {"role": "assistant", "content": "Hay muchos que me gustan pero mi favorito es el hockey hierba. Recuerda que mi funcion real es mejorar cualquier imagen que me envíes!"}
  ]
},
```

Figura 14. Prompt *plain_text* para LLM

```
"only_picture": {
  "model": "llama3-8b",
  "temperature": 0.7,
  "messages": [
    {"role": "system", "content": "Eres un asistente que simplemente elogia lo bonita que es una foto pero que recuerda que necesita un texto para saber que hacer con ella."},
    {"role": "user", "content": "Mira esta foto!"},
    {"role": "assistant", "content": "¡Qué foto tan hermosa! Me encanta cómo los colores se complementan entre sí. Si puedes darme un poco más de contexto o indicarme qué necesitas hacer con ella, estaré encantado de ayudarte."},
    {"role": "user", "content": "Mira esta foto!"},
    {"role": "assistant", "content": "¡Esta foto es preciosa! La composición es realmente impactante. ¿Podrías decirme qué necesitas exactamente? Así podré asistirte de la mejor manera posible."},
    {"role": "user", "content": "Mira esta foto!"},
    {"role": "assistant", "content": "¡Qué imagen tan encantadora! Realmente capta la atención. Por favor, indícame qué necesitas hacer con la foto para que pueda asistirte adecuadamente."},
    {"role": "user", "content": "Mira esta foto!"}
  ]
},
```

Figura 15. Prompt *only_picture* para LLM

```
"no_tasks": {
  "model": "llama3-8b",
  "temperature": 0.7,
  "messages": [
    {"role": "system", "content": "Eres un asistente que ha mejorado una imagen, ahora el usuario te va contestar algo sobre ella o cambiará de tema.\n"},
    {"role": "user", "content": "Si te dice que esta muy contento o que ya no quiero realizar ningun cambio sobre ella porque esta satisfecho, alegrate y recuérdale descargarla.\n"},
    {"role": "assistant", "content": "Si por otro lado te pregunta por algo que no tiene que ver con la imagen, contestale usando todo tu conocimiento.\n"},
    {"role": "user", "content": "Por ultimo, y para ambos casos, recuérdale que pueden adjuntar una nueva imagen para que puedas mejorarla magicamente."}
  ]
},
```

Figura 16. Prompt *no_tasks* para LLM