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’hiperparámetres 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. Problem statement and opportunity | * 1. Planteamiento del problema y oportunidad | S | 1 |
| * 1. Constraints (standards, codes, needs, requirements & specifications) | * 1. Toma en consideración de los condicionantes (normas técnicas y regulación, necesidades, requisitos y especificaciones) | S | 2 |
| * 1. Setting of goals | * 1. Establecimiento de objetivos | S | 2 |
| 1. FORMULATE: | 1. FORMULAR: |  |  |
| * 1. Creative solution generation (analysis) | * 1. Generación de soluciones creativas (análisis) |  |  |
| * 1. Evaluation of multiple solutions and decision-making (synthesis) | * 1. Evaluación de múltiples soluciones y toma de decisiones (síntesis) |  |  |
| 1. SOLVE: | 1. RESOLVER: |  |  |
| * 1. Fulfilment of goals | * 1. Evaluación del cumplimiento de objetivos |  |  |
| * 1. Overall impact and significance (contributions and practical recommendations) | * 1. Evaluación del impacto global y alcance (contribuciones y recomendaciones prácticas) |  |  |

Índice

[Capítulo 1. Introducción y objetivos 1](#_Toc170733046)

[1.1 Introducción y contexto 1](#_Toc170733047)

[1.2 Objetivos 2](#_Toc170733048)

[Capítulo 2. Metodología 3](#_Toc170733049)

[2.1 Lenguaje de programación 3](#_Toc170733050)

[2.2 Identificación y Selección de Modelos 3](#_Toc170733051)

[2.2.1 Identificación de modelos 3](#_Toc170733052)

[2.2.2 Proceso de selección de modelos 3](#_Toc170733053)

[2.3 Gestión de proyecto y control de versiones 4](#_Toc170733054)

[2.4 Estructura del proyecto 6](#_Toc170733055)

[2.5 Implementación de modelos y optimización 7](#_Toc170733056)

[2.6 Interfaz de usuario 7](#_Toc170733057)

[Capítulo 3. Selección de tecnologías 9](#_Toc170733058)

[3.1 Python 9](#_Toc170733059)

[3.2 PyCharm 9](#_Toc170733060)

[3.3 Git y GitHub 9](#_Toc170733061)

[3.4 Poetry 9](#_Toc170733062)

[3.5 Jupyter Notebooks 10](#_Toc170733063)

[3.6 PyTorch 10](#_Toc170733064)

[3.7 CUDA 10](#_Toc170733065)

[3.8 JSON 10](#_Toc170733066)

[3.9 API Calling 10](#_Toc170733067)

[3.10 Gradio 11](#_Toc170733068)

[3.11 Weight & Bias 11](#_Toc170733069)

[Capítulo 4. Desarrollo de la plataforma 12](#_Toc170733070)

[4.1 Fase 1: Selección e implantación de modelos 12](#_Toc170733071)

[4.1.1 Selección de modelos 12](#_Toc170733072)

[4.1.2 Implementación de los modelos 14](#_Toc170733073)

[4.2 Fase 2: Implementación de LLM 16](#_Toc170733074)

[4.2.1 Uso de Llama API 17](#_Toc170733075)

[4.2.2 Function calling 17](#_Toc170733076)

[4.2.3 Clase Llama 17](#_Toc170733077)

[4.3 Fase 3: Creación de la interfaz gráfica de usuario (GUI) 19](#_Toc170733078)

[4.3.1 Editor manual 19](#_Toc170733079)

[4.4 Fase 4: Entrenamiento de modelo 19](#_Toc170733080)

[4.5 Fase 5: Documentación del código 19](#_Toc170733081)

[Bibliografía 20](#_Toc170733082)

# Introducción y objetivos

## 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.

## 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 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.

# 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.

## 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.

## Identificación y Selección de Modelos

### 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

### 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.

## 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].

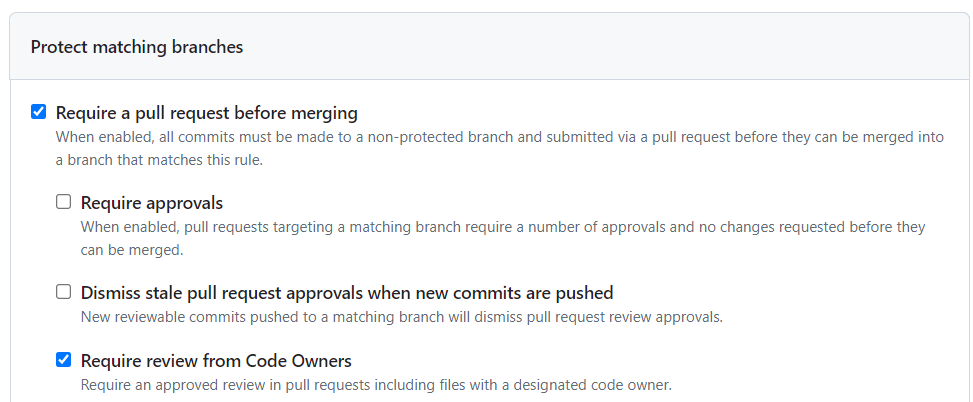
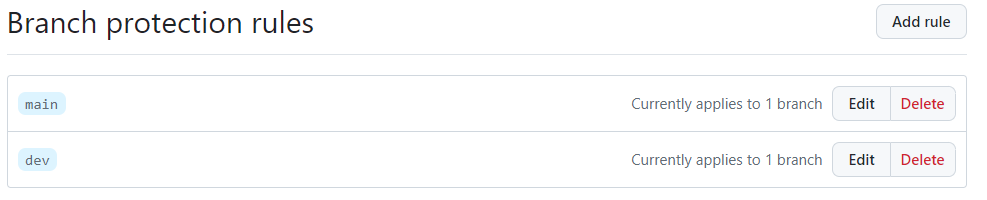


Figura .. Configuración de la protección de las ramas main y dev

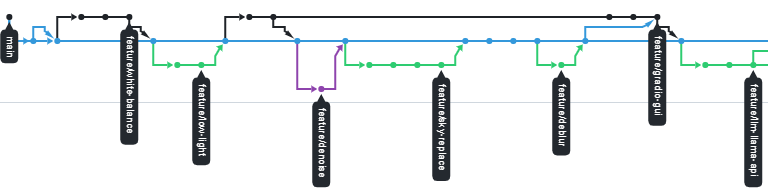


Figura .. Timeline de commits y branches en GitHub

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.

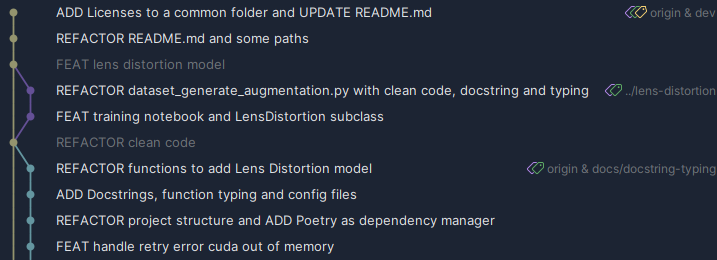


Figura .. Historial de commits



Figura .. 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.

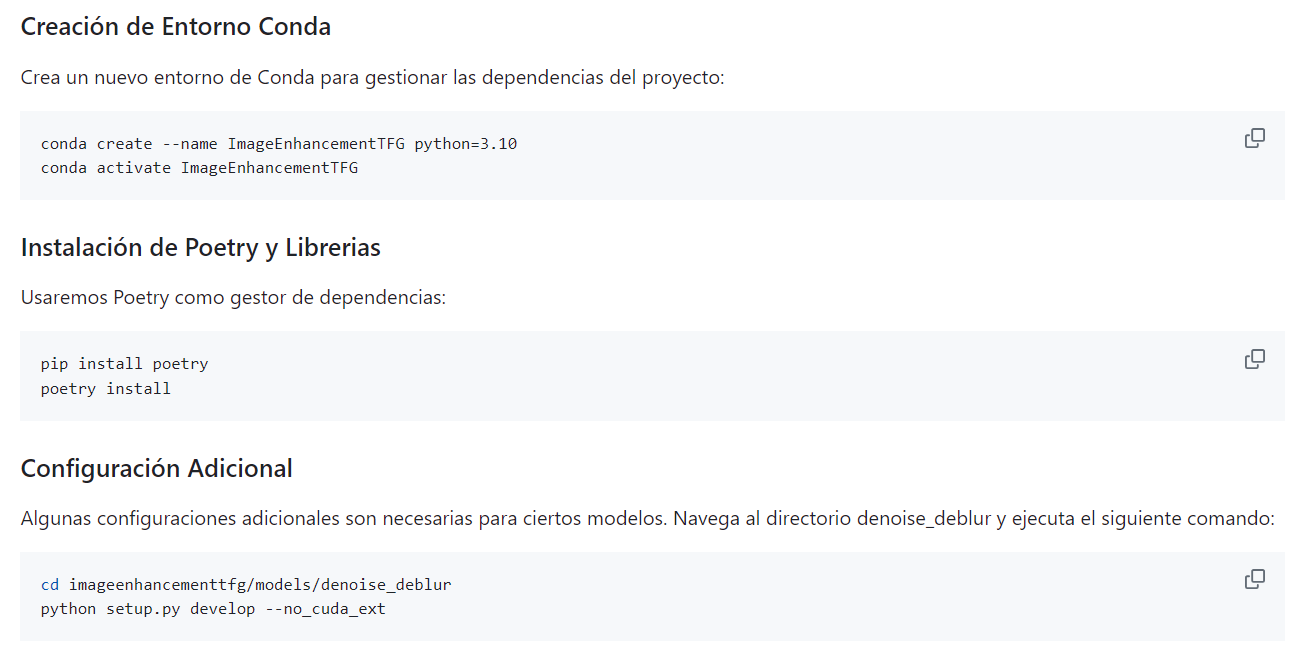


Figura .. Explicación de instalación y ejecución en el repositorio de Git

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.

## 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:

* **imageenhancementtfg**:
  + **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.

## 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.

## 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.

# 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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.

## 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 modelo 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.

## 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.

## 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.

## 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.

## 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

# Desarrollo de la plataforma

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

## Fase 1: Selección e implantación de modelos

### 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]:

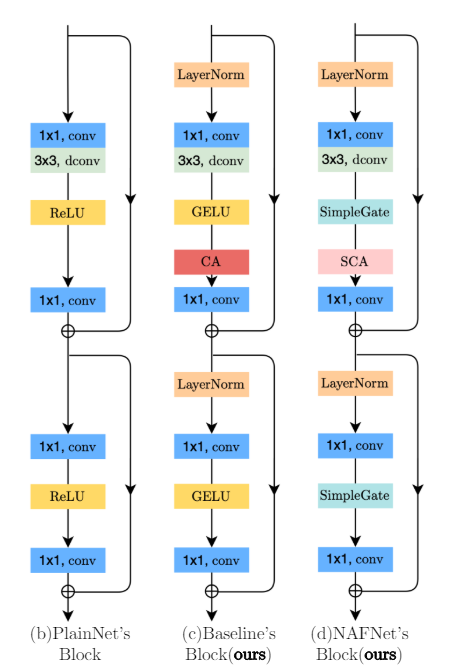


Figura .. Evolución de la arquitectura de NAFNet

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 4.7.

* **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.

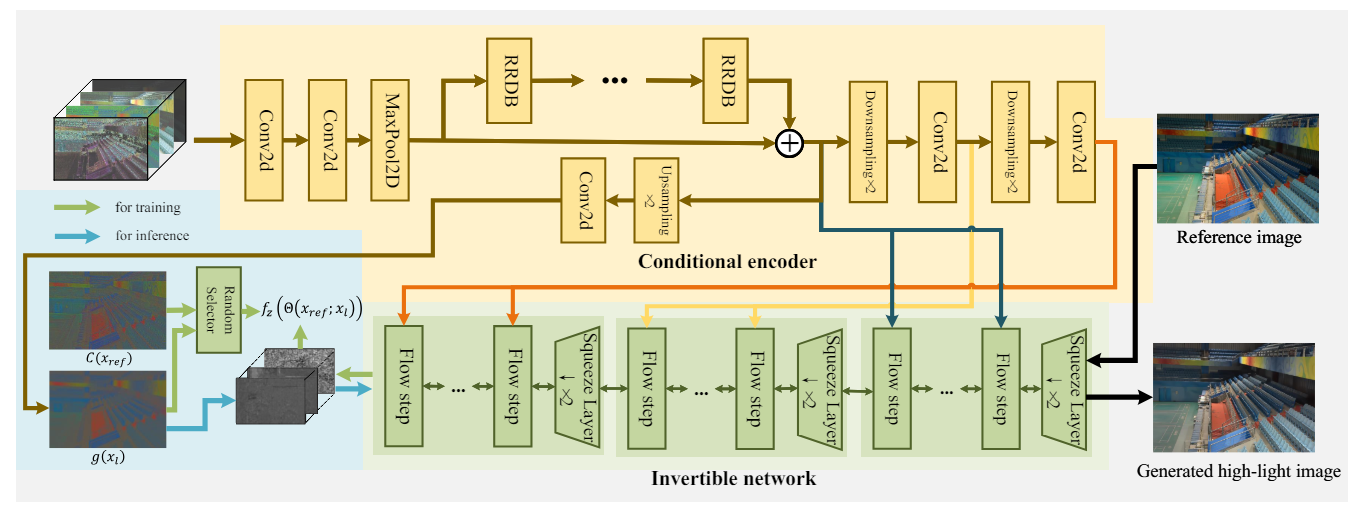


Figura .. Arquitectura de LLFlow

La implementación de esta red neuronal en mi plataforma y la estructura en su correspondiente carpeta se refleja en la Figura 4.7.

* **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. Multiples 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.

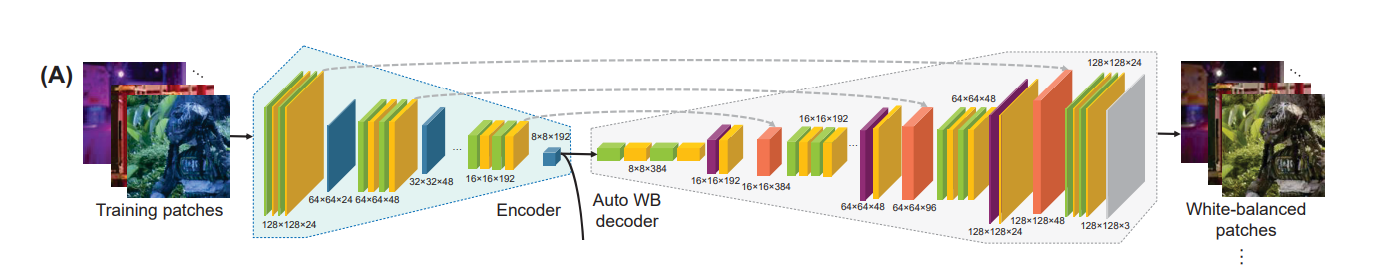


Figura .. Arquitectura del modelo con AWB

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 4.7.

* **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.

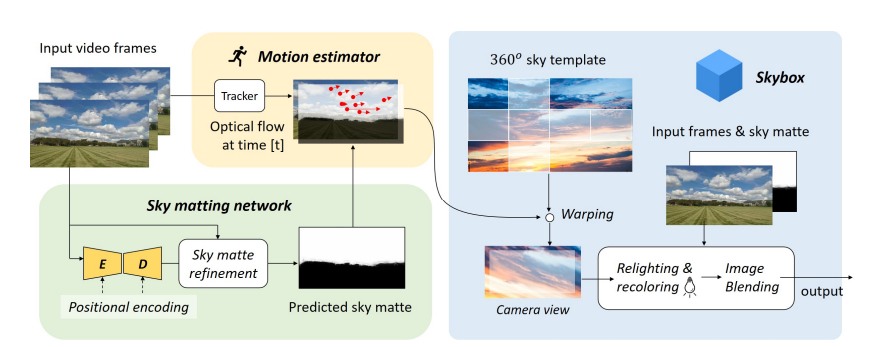


Figura .. Arquitectura de SkyAR

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 divididen 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 4.7.

### 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.

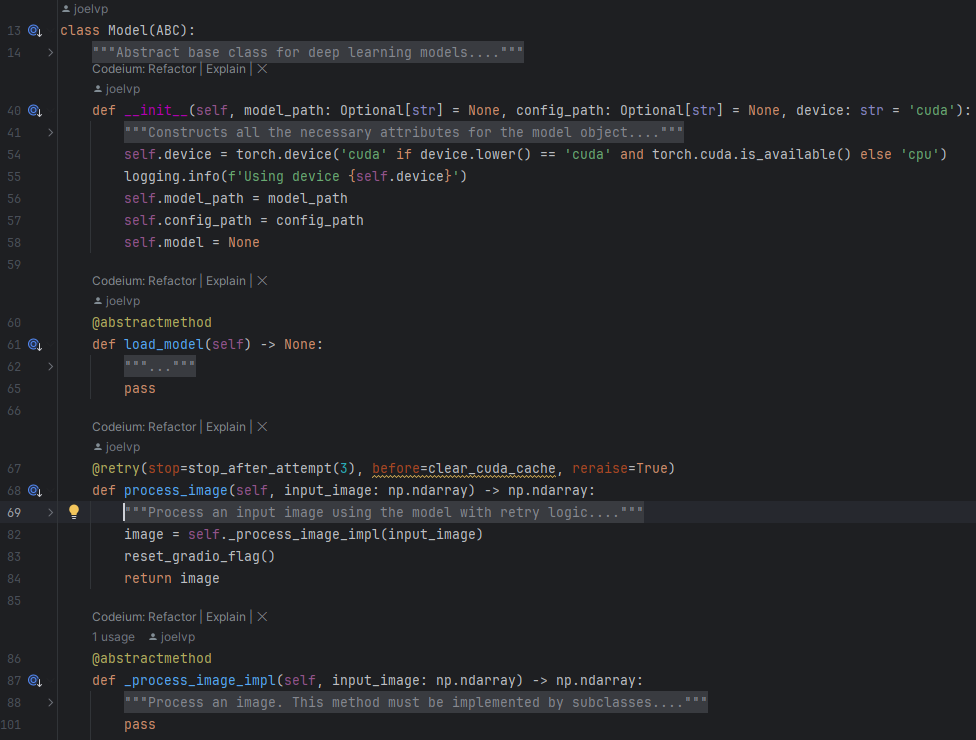


Figura .. Clase abstracta Model

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.

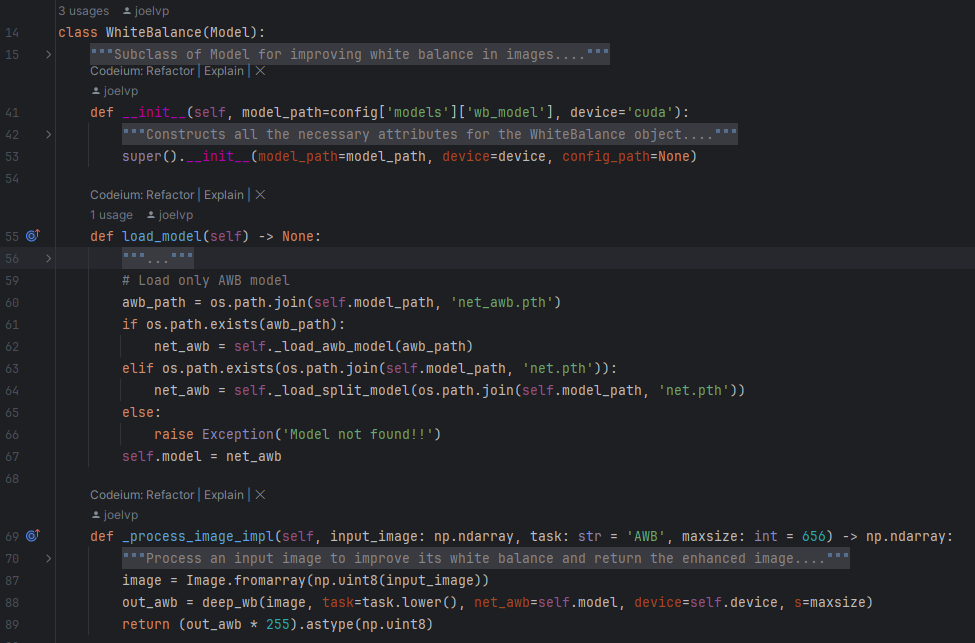


Figura .. Implementación de la clase abstracta Model en WhiteBalance

EXPLICAR CASO DEL SKYAR, YA QUE ESTABA PENSADO PARA VIDEO

Finalmente, la estructura del proyecto con los modelos implementados se refleja en la Figura 4.7.



Figura .. Estructura por carpetas del proyecto

## 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 4.8, 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 .. Principales modelos LLM

### 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.

### 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 seria imposible integrar una LLM en una plataforma de estas características.

### 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 4.9, 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.

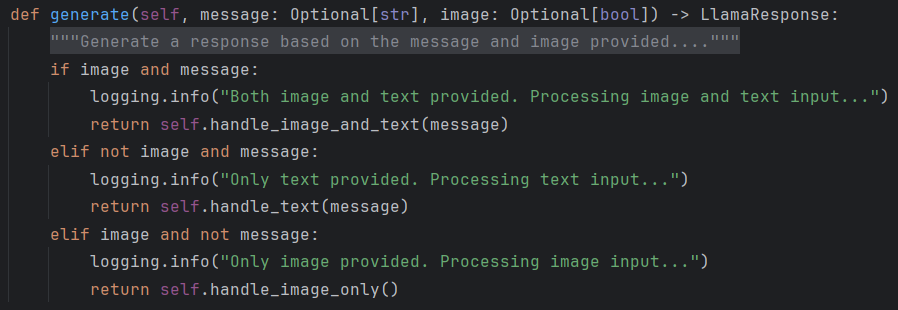


Figura .. Función principal en la generación de respuestas con Llama

* **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.

CAPTURA PROMPT?

* **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.

CAPTURA PROMPT?

* **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 salido desestructurada de la LLM a una salida concreta siguiendo nuestros parámetros, en nuestro caso devolverá un diccionario con las siguientes claves:

Figura 4.10. Salida estructura de la LLM

Para obtener este resultado usamos el siguiente prompt:

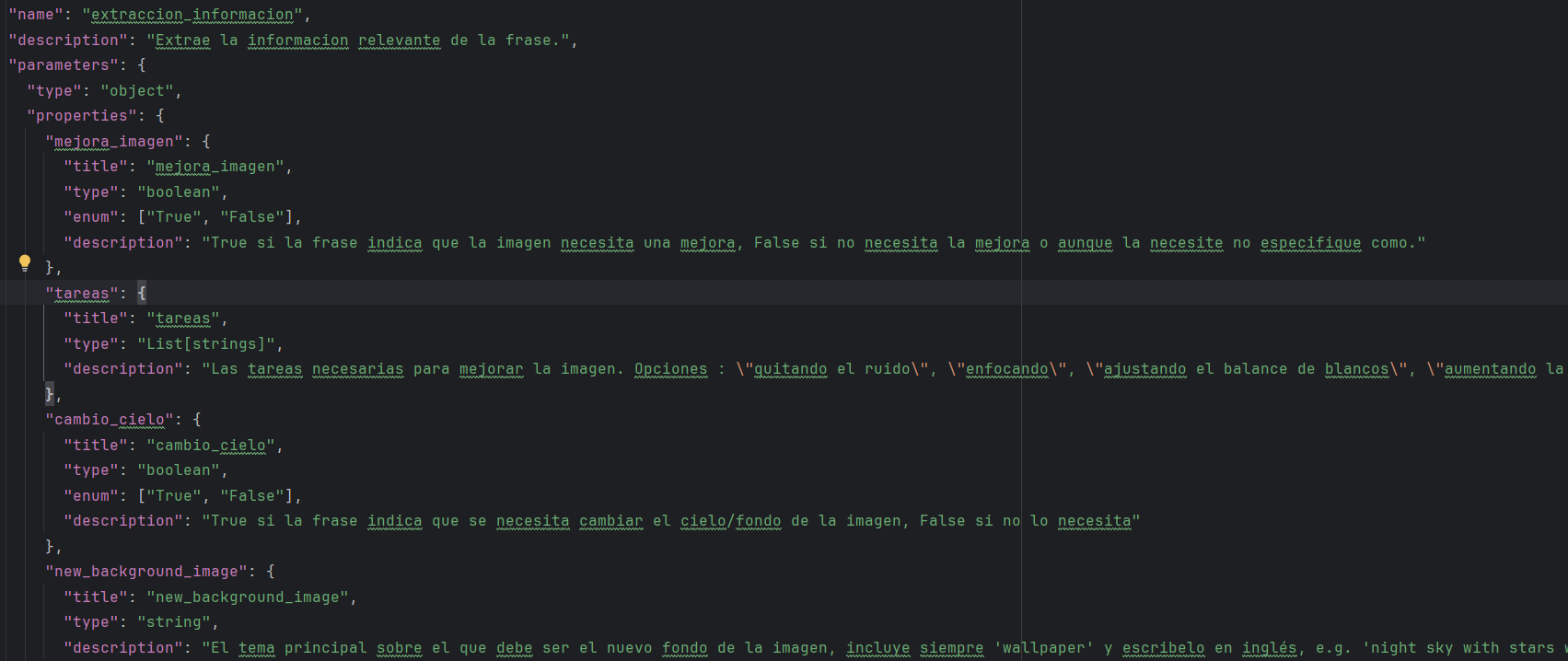
La función generate devuelve un objeto de tipo LlamaResponse, es una estructura de datos que encapsula la respuesta generada por la API. Utiliza el decorador @dataclass para simplificar su definición y la podemos observar en la Figura 4.12. Para poder alcanzar esta estructura todas las respuestas serán post procesadas y dependiendo de las condiciones que cumplan se creará de una manera o de otra.

Figura 4.11. Prompt para extraer las tareas de mejora de imagen con function calling

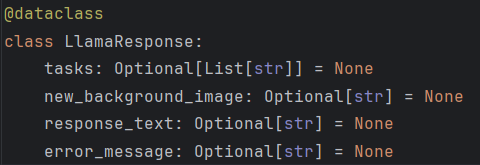


Figura .. Objeto que devolverá la función generete

## 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:

### 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 “Enchance”, se aplican los filtros seleccionados a las imágenes, cuyos resultados se muestran en una galería de salida. Muestro un ejemplo:

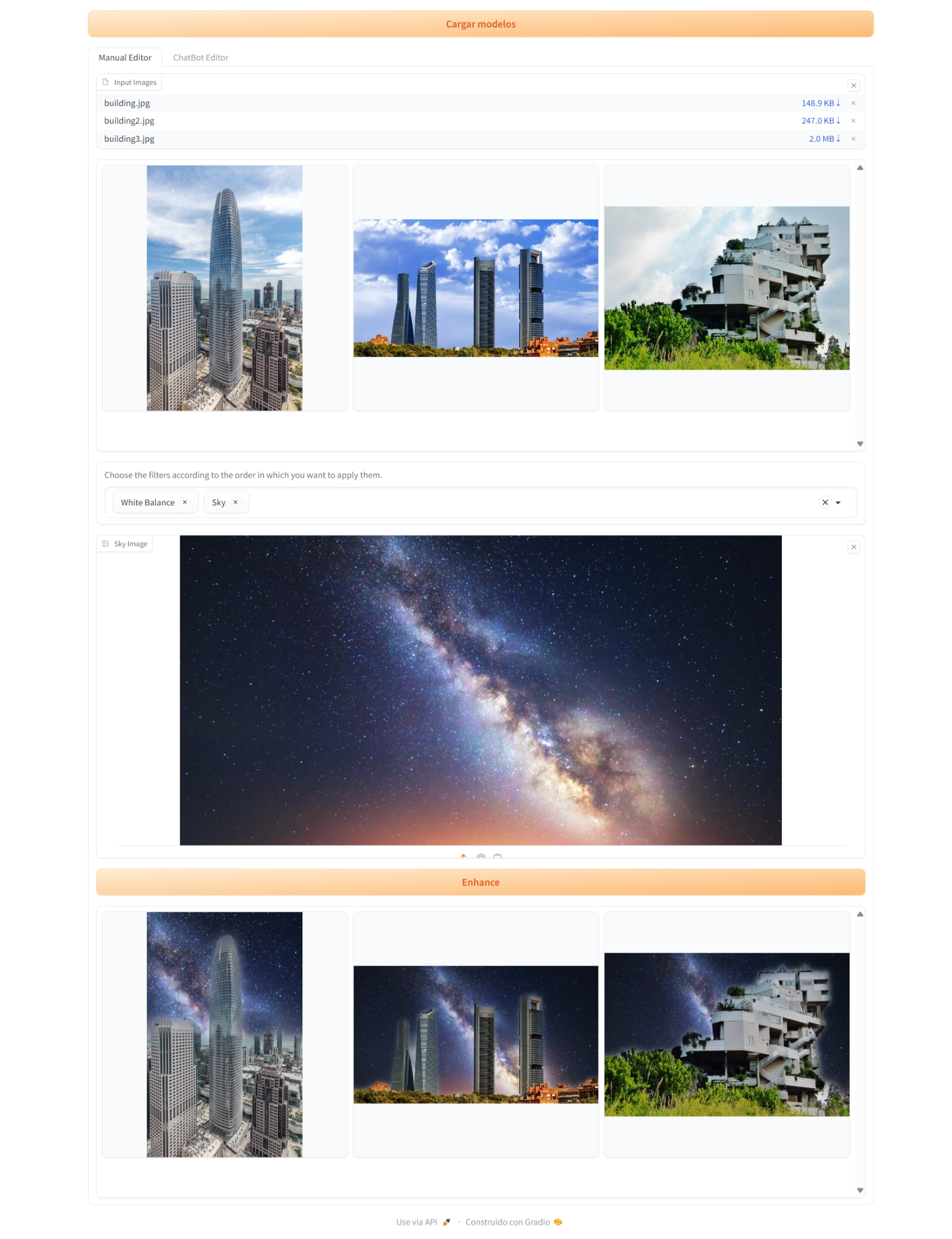


Figura .. Interfaz de usuario con editor manual

### 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 enviemos.

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:

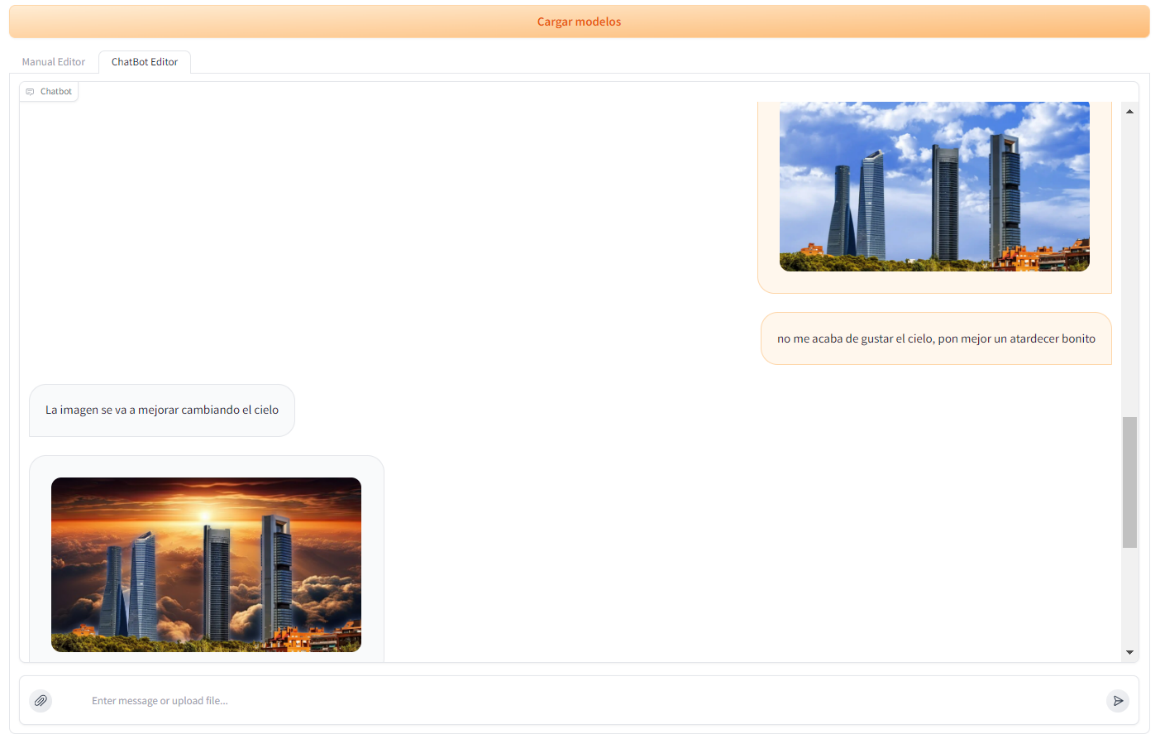


Figura .. Interfaz de usuario con LLM ChatBot

### 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 demejora 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 busqueda. 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 4.15, 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 intrepretadas 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 imágenes, para transformar imagenes en ptahs 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.

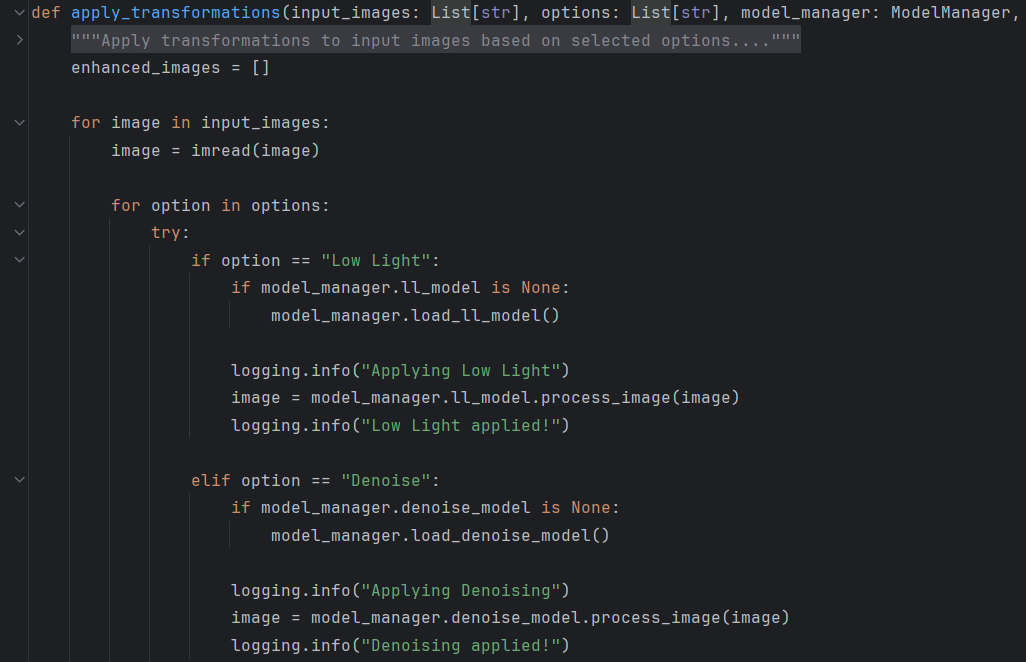


Figura .. Fragmento de la función apply\_transfromations

## Fase 4: Entrenamiento de modelo

### 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:

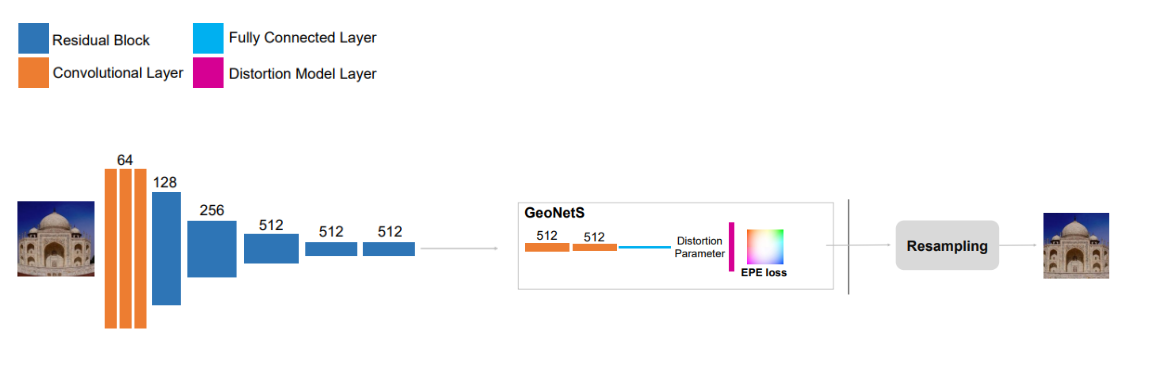
1. Extracción parámetros de distorsión: Inicialmente, se emplean tres capas convolucionales para capturar patrones visuales básicos como bordes y texturas. A continuación, se utilizan cinco bloques residuales para extraer características más complejas de manera gradual, preservando al mismo tiempo información 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 4.16. Arquitectura de GeoProj para la distorsión de lente (FishEye)

### 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 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 cumplían 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 tiene 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.

SEGUIR EXPLICACION

### Hiperparámetros y entrenamiento

### Inferencia en notebook

## Fase 5: Documentación del código

Typing and docstring

# 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,» [En línea]. Available: https://deci.ai/blog/sota-dnns-overview/. |
| [3] | V. d. P. Joel, «GitHub,» 2024. [En línea]. Available: https://github.com/joelvp/ImageEnhancementTFG. |
| [4] | A. Amin, «Medium,» 1 5 2023. [En línea]. Available: https://medium.com/@abhay.pixolo/naming-conventions-for-git-branches-a-cheatsheet-8549feca2534. |
| [5] | V. d. P. Joel, «GitHub,» 2024. [En línea]. 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. [En línea]. 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. [En línea]. 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. [En línea]. 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. [En línea]. 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. [En línea]. 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. |