

# Coloración de imágenes

Álvaro Felipe Pérez<sup>1</sup>, Miguel Gómez Prieto<sup>1</sup>, Hugo Gómez-Caraballo López-Romero<sup>1</sup> y Carlos Serrano Pinós<sup>1</sup>

<sup>1</sup>Estudiante CDIA, Universidad Politécnica de Madrid, Madrid

## Resumen

La coloración automática de imágenes en escala de grises constituye un desafío relevante en visión por computador, al requerir la generación de información cromática plausible a partir de datos incompletos. En este trabajo se presenta un análisis comparativo de diferentes arquitecturas de aprendizaje profundo aplicadas a la coloración de imágenes, incluyendo Autoencoders (AE), Variational Autoencoders (VAE), redes UNet, Generative Adversarial Networks (GAN), modelos de difusión y enfoques guiados por texto. Los experimentos se realizaron sobre el conjunto de datos STL-10 y un dataset complementario de flores, con el objetivo de evaluar la capacidad de cada modelo para reconstruir color de manera realista y coherente con el contenido semántico. Se discuten las ventajas y limitaciones de cada aproximación, así como la influencia de la resolución y el tipo de condicionamiento en la calidad perceptual de los resultados. Los hallazgos contribuyen a comprender el potencial y las fronteras actuales de las técnicas de coloración basadas en deep learning.

**Keywords:** Image colorization, Autoencoder, UNet, Generative Adversarial Network, Diffusion models, Text-guided coloration, Computer vision

# Índice

|   |           |
|---|-----------|
| <b>1. Introducción</b>  | <b>4</b>  |
| <b>2. Estudio del Estado del Arte</b>                         | <b>5</b>  |
| <b>3. Autoencoder</b>   | <b>6</b>  |
| 3.1. U-Net Autoencoder . . . . .                              | 6         |
| 3.2. Residual U-Net Autoencoder . . . . .                     | 7         |
| 3.3. Justificación de modelos . . . . .                       | 7         |
| 3.4. Entrenamiento . . . . .                                  | 8         |
| 3.5. Resultados . . . . .                                     | 8         |
| 3.6. Conclusiones . . . . .                                   | 9         |
| <b>4. VAE</b>   | <b>10</b> |
| 4.1. VAE Denso . . . . .                                      | 10        |
| 4.2. VAE Convolutional . . . . .                              | 10        |
| 4.3. Justificación de modelos . . . . .                       | 10        |
| 4.4. Entrenamiento . . . . .                                  | 11        |
| 4.5. Resultados . . . . .                                     | 11        |
| 4.6. Conclusiones . . . . .                                   | 13        |
| <b>5. GAN para coloración</b>                                 | <b>13</b> |
| 5.1. Arquitectura del Generador U-Net . . . . .               | 13        |
| 5.2. Arquitectura del Discriminador PatchGAN . . . . .        | 14        |
| 5.3. Funciones de pérdida . . . . .                           | 15        |
| 5.4. Entrenamiento . . . . .                                  | 16        |
| 5.5. Resultados . . . . .                                     | 16        |
| <b>6. GAN Guiada por Segmentación</b>                         | <b>17</b> |
| 6.1. Obtención de Máscaras Semánticas con SegFormer . . . . . | 18        |
| 6.2. Integración de la Máscara en la GAN . . . . .            | 19        |
| 6.3. Función de Pérdida . . . . .                             | 19        |
| 6.4. Entrenamiento . . . . .                                  | 20        |
| 6.5. Resultados y Métricas . . . . .                          | 20        |
| <b>7. Coloración con modelos de difusión</b>                  | <b>22</b> |
| 7.1. Arquitectura . . . . .                                   | 22        |
| 7.1.1. Entrenamiento desde cero del UNet . . . . .            | 22        |
| 7.1.2. Fine-tuning del UNet de Stable Diffusion . . . . .     | 22        |
| 7.1.3. Adaptación mediante LoRA . . . . .                     | 23        |
| 7.2. Entrenamiento . . . . .                                  | 23        |
| 7.3. Hiperparámetros . . . . .                                | 25        |
| 7.4. Resultados . . . . .                                     | 26        |

|  |           |
|--|-----------|
| <b>8. Coloración Guiada por Texto</b>      | <b>27</b> |
| 8.1. Dataset empleado . . . . .            | 28        |
| 8.2. Arquitectura del modelo . . . . .     | 28        |
| 8.2.1. Embedding textual . . . . .         | 28        |
| 8.2.2. Estructura general . . . . .        | 28        |
| 8.2.3. Bloques convolucionales . . . . .   | 29        |
| 8.2.4. Condicionamiento textual . . . . .  | 29        |
| 8.3. Entrenamiento . . . . .               | 29        |
| 8.4. Resultados . . . . .                  | 30        |
| <b>9. Conclusiones</b>                     | <b>31</b> |
| 9.1. Coloración guiada por texto . . . . . | 32        |
| <b>Referencias</b>                         | <b>33</b> |

## Introducción

La coloración automática de imágenes en escala de grises ha sido un reto clásico en el campo de la visión por computador y el procesamiento digital de imágenes. Desde sus primeras aplicaciones en la restauración, este problema plantea una dificultad fundamental: reconstruir información cromática que no está presente en los datos originales. La tarea no consiste únicamente en añadir color de manera arbitraria, sino en generar tonalidades plausibles y coherentes con la semántica de la escena, preservando al mismo tiempo la estructura y los detalles de la imagen.

Con el auge del aprendizaje profundo, la coloración ha experimentado un avance significativo. Las arquitecturas modernas permiten aprender representaciones latentes que capturan tanto la estructura como la distribución cromática de los datos, ofreciendo resultados mucho más realistas que los métodos tradicionales basados en reglas o en interpolación manual. En este contexto, diferentes enfoques han sido explorados: desde **Autoencoders (AE)** y **Variational Autoencoders (VAE)**, que aprenden a comprimir y reconstruir imágenes en espacios latentes, hasta **redes UNet**, que aprovechan conexiones de salto para preservar detalles espaciales durante la reconstrucción.

Por otro lado, las **Generative Adversarial Networks (GAN)** han demostrado un gran potencial en la generación de imágenes fotorrealistas, gracias a la dinámica competitiva entre generador y discriminador. En coloración, las GAN permiten producir colores más vivos y naturales, aunque a menudo requieren un entrenamiento cuidadoso para evitar inestabilidades. Más recientemente, los **modelos de difusión** han emergido como una alternativa poderosa, capaces de generar imágenes de alta calidad mediante un proceso iterativo de refinamiento desde ruido hacia datos estructurados. Estos modelos, al operar en espacios latentes comprimidos por un VAE, han mostrado resultados sobresalientes en tareas de síntesis y edición de imágenes, incluyendo la coloración.

Un aspecto adicional que ha cobrado relevancia es el **condicionamiento mediante texto**. La posibilidad de guiar el proceso de coloración con descripciones lingüísticas abre la puerta a aplicaciones creativas y personalizadas, donde el usuario puede especificar estilos, paletas cromáticas o incluso contextos narrativos. Aunque este enfoque introduce complejidad adicional, también amplía el rango de control sobre el resultado final.

En este trabajo se presenta un análisis comparativo de estas arquitecturas aplicadas a la coloración de imágenes, utilizando como base el conjunto de datos **STL-10**, ampliamente empleado en tareas de clasificación y representación visual, y un dataset complementario de flores, que aporta diversidad cromática y riqueza semántica. Además, se explora la hipótesis de que una **segmentación previa a la coloración** puede mejorar la coherencia de los resultados, al proporcionar información estructural más precisa sobre los objetos presentes en la escena.

La introducción de este estudio busca situar la coloración automática dentro del panorama actual de la inteligencia artificial aplicada a la visión por computador, destacando tanto su relevancia práctica como su interés científico. A través de la comparación de AE, VAE, UNet, GAN, modelos de difusión y enfoques guiados por texto, junto con la incorporación de segmentación como paso previo, se pretende ofrecer una visión integral de las técnicas más representativas y de su impacto en la calidad perceptual de las imágenes generadas.

## Estudio del Estado del Arte

Los avances recientes en coloración automática de imágenes se estructuran en diversas familias metodológicas que definen el estado del arte en esta área. En primer lugar, los métodos basados en redes convolucionales continúan siendo un pilar fundamental dentro del aprendizaje profundo. Estas aproximaciones suelen apoyarse en variantes refinadas de arquitecturas tipo U-Net, donde se integran convoluciones dilatadas, técnicas de normalización y diversas modificaciones estructurales destinadas a mejorar la preservación del detalle y la coherencia espacial.

En paralelo, los modelos sustentados en mecanismos de atención y arquitecturas Transformer han experimentado un crecimiento notable, impulsados por su capacidad para modelar dependencias de largo alcance entre regiones distantes de la imagen. Esta propiedad se traduce habitualmente en una mayor consistencia cromática y semántica, especialmente en escenas complejas donde la comprensión global del contexto resulta determinante.

Otra línea destacada la constituyen los métodos basados en redes GAN, junto con las variantes híbridas que combinan GANs y módulos de atención. Estos sistemas han demostrado una capacidad superior para producir coloraciones naturales y vívidas, mitigando problemas habituales como la desaturación cromática.

Asimismo, las arquitecturas híbridas con *dual-decoder* representan una contribución relevante al buscar un equilibrio óptimo entre fidelidad local y coherencia global. Este tipo de diseños permite reducir artefactos característicos del proceso de coloración, como el *color bleeding*, al separar explícitamente la reconstrucción estructural del razonamiento semántico.

De forma complementaria, los estudios de carácter *survey* y los trabajos de *benchmarking* publicados en los últimos años proporcionan una visión integradora del campo. Estos análisis comparativos revisan las limitaciones de los conjuntos de datos disponibles, evidencian la presencia de sesgos cromáticos y discuten la necesidad de métricas más representativas para evaluar adecuadamente la naturalidad, la coherencia semántica y la diversidad cromática de los modelos actuales.

Entre los modelos contemporáneos más representativos destaca DDColor, considerado uno de los enfoques con mejor equilibrio entre estructura y semántica gracias a su arquitectura basada en dos decodificadores complementarios:

- Pixel Decoder (Estructural): Responsable de reconstruir la resolución espacial, las texturas y la definición de bordes.
- Color Decoder (Semántico): Basado en tecnología Transformer y *Color Queries*, permitiendo asignar colores coherentes y vibrantes en función del contenido semántico, sin comprometer los límites estructurales.

La combinación de ambas ramas permite obtener coloraciones altamente detalladas y cromáticamente realistas, superando a numerosos métodos previos basados exclusivamente en GANs [1].

Si el objetivo es maximizar el realismo visual, uno de los modelos más avanzados es CtrlColor,

basado en técnicas de difusión. Este método mejora significativamente las limitaciones presentes en trabajos previos, especialmente en relación con el *color bleeding*. Su principal aportación consiste en un marco unificado que integra tres modos de operación dentro de una única red, sin necesidad de reentrenamiento:

- Incondicional: Permite la coloración automática al estilo de DDColor.
- Guiado por texto: Utiliza prompts para orientar la paleta cromática.
- Guiado por trazos: El usuario puede especificar trazos de color, que el modelo propaga respetando la estructura geométrica de la imagen.

Esta versatilidad es posible gracias a una arquitectura de difusión latente que incorpora un Codificador de Contenido especializado. Dicho codificador actúa como una restricción estructural explícita, garantizando que los bordes y formas originales limiten el proceso generativo y evitando que la difusión genere colores fuera de las regiones correspondientes [2].

Finalmente, la vanguardia actual apunta hacia modelos basados en Flow Matching, entre los que destaca ModFlows. Este enfoque marca una transición conceptual desde los métodos de difusión hacia los denominados *flujos rectificados*. Su principal innovación radica en abandonar el proceso estocástico y progresivo de eliminación de ruido característico de la difusión, sustituyéndolo por una trayectoria determinista que transporta directamente la distribución de color desde una imagen de referencia hasta la imagen objetivo. Este cambio permite acelerar drásticamente la inferencia, entre 20 y 30 veces respecto a los modelos de difusión, y garantiza resultados más estables, coherentes y libres de alucinaciones [3].

## Autoencoder

Comenzaremos nuestro proyecto probando ciertos modelos sencillos para ver su desempeño sobre el dataset STL-10 [4]. En este apartado hablaremos sobre dos diferentes tipos de autoencoders cuya única entrada es una imagen en blanco y negro y su salida es la imagen reconstruida a color.

### U-Net Autoencoder

El modelo implementado sigue una arquitectura tipo U-Net simétrica [5]. A diferencia de un autoencoder secuencial estándar, esta red incorpora conexiones de salto (skip connections) que unen las capas del codificador con sus correspondientes en el decodificador. La arquitectura se compone de tres bloques funcionales detallados a continuación:

- Encoder: recibe un tensor de dimensiones (96x96x1). Consta de 3 bloques secuenciales. Cada bloque aplica una convolución de 3x3 con activación ReLU y padding "same" para mantener las dimensiones, seguida inmediatamente de una operación de Max Pooling que reduce el tamaño espacial a la mitad.
- Cuello de botella: comprime al máximo la información (12x12). Se utiliza una capa convolucional de 256 filtros para conectar ambos bloques.

- Decoder: aplicamos UpSampling2D para duplicar el tamaño de la imagen cada vez. Uno de sus detalles más importantes es que se concatena información con capas del encoder permitiendo al modelo recordar la ubicación exacta de los bordes y texturas que suelen perderse durante la compresión del Max Pooling.  
Finalmente, se utiliza una capa convolucional con activación sigmoide que proyecta el resultado a los tres canales R, G, B.

## Residual U-Net Autoencoder

Este segundo modelo propuesto sustituye las convoluciones estándar por Bloques Residuales e introduce mecanismos de normalización y submuestreo aprendible [6]. A continuación se detallan sus componentes clave:

- Bloques residuales: Esta es la unidad fundamental de la red. A diferencia de una capa convolucional simple que intenta aprender una función  $H(x)$  directamente, este bloque intenta aprender el residuo  $F(x)$ , de tal forma que la salida sea  $H(x)=F(x)+x$ .
- Encoder: similar al anterior
- Decoder: se diferencia del anterior porque utiliza capas deconvolucionales en lugar de capas upsampling. Las capas deconvolucionales tienen pesos entrenables. La red aprende la mejor manera de aumentar la resolución de la imagen, recuperando texturas y bordes de forma más inteligente que una simple interpolación bilineal.  
Finalmente, se utiliza una capa convolucional con activación sigmoide que proyecta el resultado a los tres canales R, G, B.

## Justificación de modelos

Se descarta el uso de autoencoders secuenciales simples debido a que el proceso de compresión provoca una pérdida irreversible de información espacial. Esto provoca el fenómeno de color bleeding, donde el color se desborda de los contornos al no existir una referencia clara de los bordes durante la reconstrucción.

Por el contrario, la elección de las arquitecturas U-Net y ResU-Net se fundamenta en su capacidad para resolver este compromiso:

- Recuperación Espacial: Las skip connections reintroducen los detalles de alta resolución bordes y texturas directamente desde la entrada, garantizando que el color se aplique con cierta precisión sobre la estructura original.
- Capacidad de Aprendizaje: La inclusión de bloques residuales permite aumentar la profundidad de la red sin degradar el entrenamiento, facultando al modelo para aprender relaciones de color complejas y sutiles que un modelo superficial no podría captar.

## Entrenamiento

Ambos modelos se compilan utilizando el optimizador Adam y la función de pérdida MSE, estándar para regresión de píxeles. Sin embargo, para la ResU-Net se ha reducido el learning rate a 0.0005 para garantizar la estabilidad del gradiente dada su mayor profundidad.

Además, es interesante comentar la diferencia significativa en la carga computacional debido a la complejidad de las arquitecturas:

- Modelo U-Net: menos de 2 min/epoch. Al ser más ligero, permite un entrenamiento más extenso (40 epochs).
- Modelo ResU-Net: sobre 8 min/epoch. La inclusión de bloques residuales, normalización por lotes y mayor número de filtros (hasta 512) incrementa drásticamente las operaciones por ciclo, lo que ha obligado a limitar el entrenamiento.

Para acelerar el entrenamiento de ambos modelos se ha implementado una política de Mixed Precision (FP16).

## Resultados

Mostramos en la siguiente tabla las métricas obtenidas en test de ambos modelos:

Cuadro 1: Comparación de métricas entre U-Net y ResU-Net

| Métrica   | U-Net   | ResU-Net |
|-----------|---------|----------|
| SSIM      | 0.9323  | 0.9360   |
| PSNR      | 25.3312 | 25.0441  |
| LPIPS     | 0.1345  | 0.1466   |
| CIEDE2000 | 8.7632  | 8.8882   |

Los resultados cuantitativos muestran un rendimiento muy parejo entre ambos modelos, con ligeras variaciones atribuibles a la diferencia en el tiempo de entrenamiento (40 épocas vs 5 épocas):

- **SSIM (estructura):** La ResU-Net (0.9360) supera ligeramente a la U-Net. Esto confirma que los bloques residuales y la mayor capacidad de la red ayudan a reconstruir mejor la estructura y los bordes de la imagen, incluso con menos entrenamiento.
- **PSNR y LPIPS (fidelidad y percepción):** La U-Net obtiene mejores puntuaciones (mayor PSNR, menor LPIPS). Esto se debe a que, al haber entrenado durante 40 épocas, ha tenido tiempo de converger hacia una solución más estable y minimizar el error medio, mientras que la ResU-Net (con solo 5 épocas) quizás aún no haya alcanzado su potencial máximo.

Observamos a continuación ciertas imágenes:

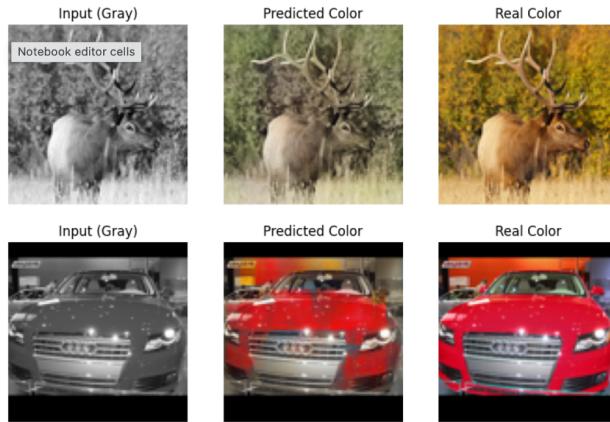


Figura 1: Imágenes generadas con U-Net

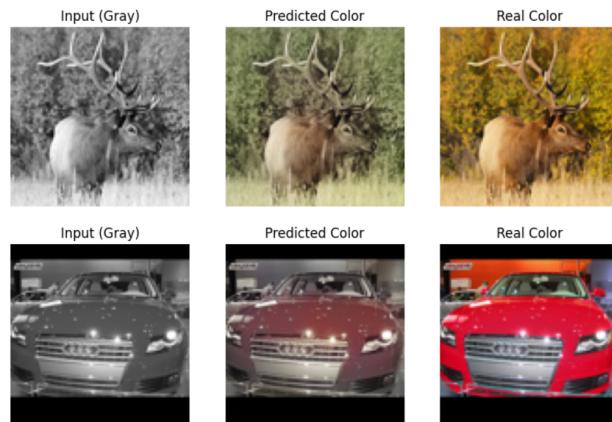


Figura 2: Imágenes generadas con ResU-Net

Vemos una mayor vivacidad y acierto con los colores de la U-Net que corresponde con los mejores resultados obtenidos en PSNR y LPIPS, pero vemos que su estructura es algo peor porque sus bordes coloreados son algo desacertados.

## Conclusiones

El análisis comparativo entre las dos arquitecturas propuestas revela un claro compromiso entre complejidad y rendimiento.

Si bien la ResU-Net demostró una superioridad estructural latente (mejor SSIM) gracias a sus bloques residuales, su elevado coste computacional limitó la convergencia cromática en comparación con la U-Net, que logró mayor fidelidad de color gracias a un entrenamiento más extenso.

A pesar de las limitaciones intrínsecas del modelo autoencoder y la pérdida MSE, que tienden a generar tonos desaturados ante la incertidumbre, los resultados cualitativos han superado las expectativas iniciales para modelos de esta complejidad,

## VAE

En este apartado, probaremos dos diferentes VAE sobre el mismo dataset y cuya entrada y salida es la misma que los autoencoders anteriores.

### VAE Denso

Este primer modelo implementa un VAE clásico con un cuello de botella denso [7]:

- Encoder: Rompe la estructura espacial de la imagen. Tras las convoluciones, utiliza Flatten para convertir el mapa de características 2D en un vector plano 1D, comprimiéndolo todo mediante capas Dense.
- Latent Space: Es un vector de números, sin altura ni anchura.
- Decoder: Debe reconstruir.<sup>el</sup> espacio desde cero. Utiliza una capa densa para expandir el vector y luego Reshape para forzarlo a tener forma de imagen antes de aplicar las deconvoluciones.

### VAE Convolucional

Este modelo implementa un Autoencoder Variacional que mantiene la topología 2D de la imagen a lo largo de todo el proceso, evitando las capas densas que destruyen la información espacial. Sus componentes técnicos clave son:

- Entrenamiento Estocástico: Implementa el Truco de Reparametrización

$$z = \mu + \sigma \cdot \epsilon$$

separando la aleatoriedad para permitir el cálculo de gradientes y el entrenamiento de la red.

- Encoder Aprendible: Sustituye el MaxPooling fijo por Convoluciones con Stride, otorgando a la red la capacidad de aprender cómo reducir la dimensión sin perder información estructural crítica. Utiliza activación LeakyReLU para garantizar la estabilidad del flujo de gradientes.
- Reconstrucción: El decodificador recupera la resolución original mediante Convoluciones Transpuestas, proyectando los detalles de color directamente desde los mapas de características espaciales del cuello de botella.

### Justificación de modelos

Estos modelos se implementan para abordar la ambigüedad intrínseca de la coloración porque un objeto gris puede tener múltiples colores válidos.

Mientras que los modelos deterministas tienden a promediar colores generando tonos apagados ante la duda, los VAEs aprenden una distribución de probabilidad, permitiendo generar colores más vivos y diversos.

Además, la creación del modelo VAE convolucional se justifica porque al mantener un espacio latente tensorial no aplanado, preserva la correlación espacial de los píxeles, resultando en una reconstrucción geométrica superior.

## Entrenamiento

Ambos modelos se basan en dos términos fundamentales en su función de pérdida: un término de reconstrucción y un término de regularización mediante divergencia KL. Sin embargo, en cada modelo lo implementamos de diferente forma

El primer modelo, utiliza MSE para medir la discrepancia entre la imagen reconstruida y la imagen real en color. Tras observar los resultados de este modelo, decidimos implementar MAE en el segundo modelo porque penaliza de forma lineal los errores, lo que tiende a producir reconstrucciones más nítidas.

Por otra parte, ambos modelos emplean la misma expresión para la divergencia KL. Sin embargo, en el primer modelo la KL se añade directamente a la pérdida total porque es un espacio latente relativamente pequeño.

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \mathcal{L}_{\text{KL}}$$

Esto ayuda a que la distribución latente sea estable y bien organizada.

En el segundo modelo la KL se multiplica por un peso  $\lambda$  muy pequeño.

$$\mathcal{L} = \mathcal{L}_{\text{rec}} + \lambda \cdot \mathcal{L}_{\text{KL}}$$

La razón es que el espacio latente ahora es mucho más grande. Si se utilizara la KL sin ponderación, esta dominaría por completo la pérdida, impidiendo que el modelo aprenda a reconstruir correctamente.

El tiempo de entrenamiento de ambos modelos es muy similar (2 min/epoch). Para estos modelos se ha desactivado Mixed Precision por sencillez.

## Resultados

Mostramos en la siguiente tabla las métricas obtenidas en test de ambos modelos:

Cuadro 2: Comparación de métricas entre VAE Denso y VAE Convolucional

| Métrica    | VAE Denso | VAE Convolucional |
|------------|-----------|-------------------|
| SSIM       | 0.0600    | 0.9338            |
| PSNR       | 6.3406    | 25.1147           |
| LPIPS      | 0.6884    | 0.1485            |
| CIEDE 2000 | –         | 8.8405            |

La comparación cuantitativa evidencia una diferencia abismal en el rendimiento de ambas arquitecturas, validando la importancia crítica de la preservación espacial en el espacio latente:

Los valores obtenidos (SSIM: 0.06, PSNR: 6.34) indican un colapso estructural casi total del VAE Denso. Al aplanar los mapas de características y utilizar capas densas, el modelo pierde la correlación espacial de los píxeles, resultando en una salida que no logra reconstruir ni siquiera la forma de los objetos, comportándose prácticamente como ruido aleatorio.

Por el contrario, el modelo totalmente convolucional alcanza un SSIM de 0.9338, situándose al mismo nivel de calidad que los modelos autoencoder. Esto demuestra que mantener la topología 2D en el cuello de botella es un requisito indispensable para la coloración.

La métrica perceptual confirma lo anterior (LPIPS): el VAE Denso (0.6884) genera imágenes irreconocibles para el ojo humano, mientras que el CVAE (0.1485) produce resultados coherentes y naturales.

Con una inspección visual observamos reflejados los resultados obtenidos cuantitativamente por las métricas:

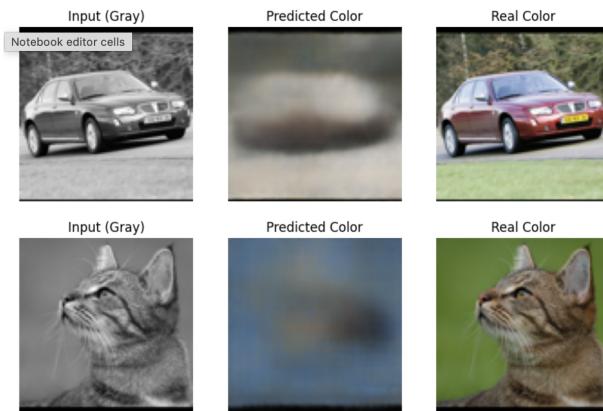


Figura 3: Imágenes generadas con VAE Denso

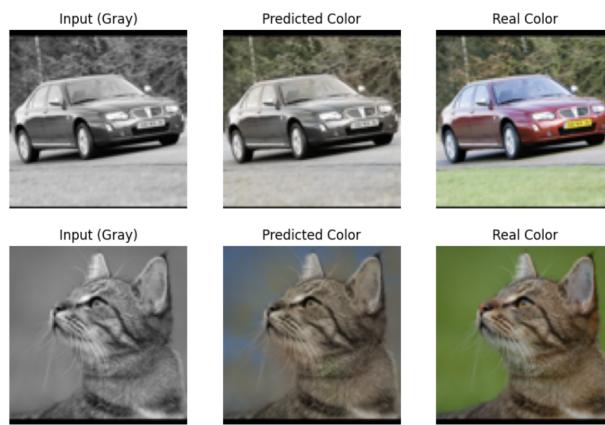


Figura 4: Imágenes generadas con VAE Convolucional

Observamos una imagen muy borrosa del primer VAE indicando que no ha sido capaz de recons-

truir la imagen. Con el segundo VAE sí tenemos una reconstrucción óptima de la imagen pero la coloración no es correcta.

## Conclusiones

La experimentación con modelos variacionales evidencia que la preservación de la topología en el espacio latente es un requisito indispensable, habiendo logrado el VAE convolucional rescatar la coherencia estructural tras el colapso del enfoque denso. Sin embargo, pese a esta validación arquitectónica, es necesario señalar que los resultados visuales finales no logran superar la calidad obtenida por los autoencoders. La restricción de regularización impuesta por la Divergencia KL, sumada a la optimización de pérdidas píxel a píxel, tiende a producir imágenes con texturas suavizadas y menor definición cromática. Esta limitación perceptiva justifica el siguiente paso en la investigación: la transición hacia las Redes GAN, cuyo mecanismo de discriminación busca romper precisamente esta barrera de borrosidad para forzar la generación de detalles de alta frecuencia y un realismo visual superior.

## GAN para coloración

Tras evaluar modelos deterministas como autoencoders y VAEs, ahora vamos a explorar un enfoque basado en redes GAN para la coloración. Las GAN son especialmente adecuadas para esta tarea porque incorporan un discriminador que empuja al generador a producir colores más realistas y coherentes, incluso en zonas donde la elección del color es más ambigua. Esto suele traducirse en imágenes más saturadas, con transiciones suaves y menos aspecto borroso que en los métodos puramente reconstrucción.

Nuestro modelo sigue la filosofía de *pix2pix*[8], formulando la coloración como un problema de *image-to-image translation*: dado el canal L de luminancia de una imagen en espacio Lab, el objetivo es predecir los canales a y b que contienen la información cromática.

## Arquitectura del Generador U-Net

Para el generador empleamos una U-Net [5], debido a su capacidad para preservar la información espacial de la imagen gracias a sus skip conexions entre encoder y decoder. En coloración esto es crucial porque aunque el color pueda variar, los bordes, contornos y texturas deben mantenerse.

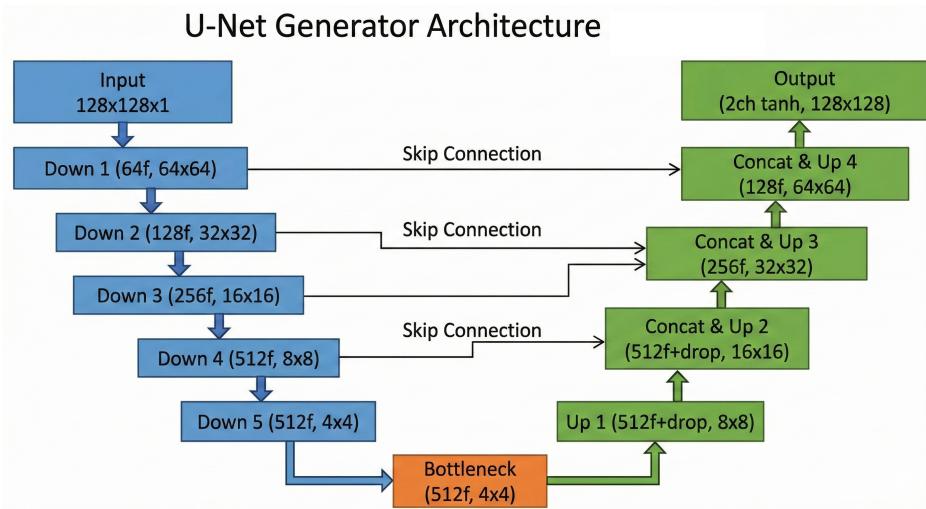


Figura 5: Diagrama de la arquitectura U-Net

El generador recibe como entrada un tensor de dimensiones  $(128 \times 128 \times 1)$  con la imagen en escala de grises. Su estructura se organiza en dos partes:

- **Bloques de downsampling:** cinco niveles de convoluciones  $4 \times 4$  con *stride* 2, que van reduciendo la resolución mientras aumentan los filtros (64, 128, 256, 512, 512). Todas las capas salvo la primera incluyen BatchNorm. Se emplea activación LeakyReLU.
- **Bloques de upsampling:** cuatro capas transpuestas de convolución para recuperar la resolución. Los dos primeros bloques incorporan *dropout* (0.5), lo que introduce variabilidad para evitar que el generador memorice patrones del entrenamiento y se dé sobreajuste.
- **Skip conexions:** cada bloque de upsampling concatena su salida con el mapa correspondiente del encoder. Esto permite reconstruir detalles finos, evitando que se pierda información por el downsampling.
- **Salida:** una convolución transpuesta adicional produce una imagen  $(128 \times 128 \times 2)$  con activación *tanh*, que corresponde a los canales a y b normalizados [-1,1].

En conjunto, la U-Net hace que las coloraciones sean más nítidas, evitando que la red rellene grandes áreas con colores uniformes o poco detallados.

## Arquitectura del Discriminador PatchGAN

Para el discriminador vamos a utilizar una variante de *PatchGAN* al igual que se hace en *pix2pix*[8], que en lugar de emitir una única predicción global de la imagen, produce un mapa de activación donde cada celda evalúa si el parche correspondiente parece real o generado.

Una particularidad de nuestro modelo es que el discriminador, además del mapa de activación de real/falso, devuelve características intermedias de varias capas. Estas representaciones se emplean

para calcular una *Feature Matching Loss*, que hace que el entrenamiento sea más estable y ayuda al generador hacia distribuciones más coherentes. Esta técnica fue introducida por Salimans et al. [9] y ayuda mejorar la calidad perceptual de la GAN, obligando al generador a reproducir las estadísticas internas del discriminador en vez de centrarse únicamente en engañarlo, esto nos va a ayudar a que los colores sean más vivos y no se dé mode-collapse.

Su estructura es la siguiente:

- Recibe la imagen de entrada L y la imagen objetivo o generada ab, formando un tensor  $(128 \times 128 \times 3)$ .
- Aplica tres bloques de convolución  $4 \times 4$  con *stride* 2, aumentando progresivamente el número de filtros (64, 128, 256).
- Añade dos convoluciones adicionales sin reducir la resolución.
- Produce un mapa final  $(14 \times 14 \times 1)$  con las predicciones por parche.
- Devuelve además las activaciones de varias capas internas para la feature loss.

Esto nos permite capturar errores que se dan forma local en ciertas partes de la imagen, que una discriminador global podría pasar por alto.

## Funciones de pérdida

La pérdida total del generador combina tres términos:

- **Pérdida adversaria** ( $\mathcal{L}_{GAN}$ ): empuja al generador a producir colores que el discriminador no pueda distinguir de los reales.
- **Pérdida L1** ( $\mathcal{L}_{L1}$ ): calcula el error absoluto medio entre los canales de color reales y generados. Favorece soluciones estables y evita saltos o colores irreales.
- **Feature Matching Loss** ( $\mathcal{L}_{FM}$ ): mide la diferencia entre las activaciones intermedias que el discriminador produce para imágenes reales y generadas. Esto actúa como una regularización y reduce el mode-collapse.

El coste total del generador es:

$$\mathcal{L}_G = \mathcal{L}_{GAN} + \lambda \mathcal{L}_{L1} + \lambda \mathcal{L}_{FM},$$

empleando  $\lambda = 10$ , un valor que balancea adecuadamente la reconstrucción y la calidad perceptual.

El discriminador minimiza:

$$\mathcal{L}_D = \mathcal{L}_{real} + \mathcal{L}_{fake}.$$

Ambas pérdidas se implementan mediante *binary cross-entropy* con logits.

## Entrenamiento

Entrenamos tanto el generador como el discriminador con Adam, usando una tasa de aprendizaje de  $2 \cdot 10^{-4}$  y  $\beta_1 = 0,5$ , siguiendo las recomendaciones clásicas en GANs.

Durante cada iteración del entrenamiento:

1. El generador predice los canales de color a partir del canal L.
2. El discriminador evalúa tanto la pareja real ( $L + ab$ ) como la generada ( $L + \hat{ab}$ ).
3. Se extraen activaciones intermedias del discriminador para calcular  $\mathcal{L}_{FM}$ .
4. Se calculan las pérdidas y se actualizan ambos modelos.

Entrenamos el modelo durante 50 epochs utilizando el dataset STL al igual que en los modelos anteriores pero tomando un subset de 20.000 imágenes unlabeled para el entrenamiento, 1.000 para validación y 8.000 para test. Todas las imágenes se redimensionaron a  $128 \times 128$  y se transformaron al espacio Lab usando *skimage*.

## Resultados

Para evaluar el rendimiento de la GAN clásica utilizamos las tres métricas anteriores: SSIM, que mide la coherencia estructural; PSNR, que cuantifica la fidelidad de reconstrucción, y LPIPS, que evalúa la similitud perceptual.

Cuadro 3: Métricas de evaluación de la GAN

| Métrica | GAN     |
|---------|---------|
| SSIM    | 0.9097  |
| PSNR    | 23.6826 |
| LPIPS   | 0.1628  |

Las métricas reflejan que el modelo es capaz de preservar adecuadamente la estructura global de la imagen (SSIM = 0.91), aunque todavía existe un error de reconstrucción apreciable (PSNR = 23.7), como es de esperar por la dificultad inherente de la coloración. El valor de LPIPS (0,1628) indica que, a nivel perceptual, las predicciones son razonablemente cercanas a las imágenes reales, pero aún presentan errores en la coloración de zonas más ambiguas.

En la Figura 6 podemos ver algunos de los ejemplos del conjunto de test, comparando las imágenes originales en escala de grises con las generadas y la original a color.

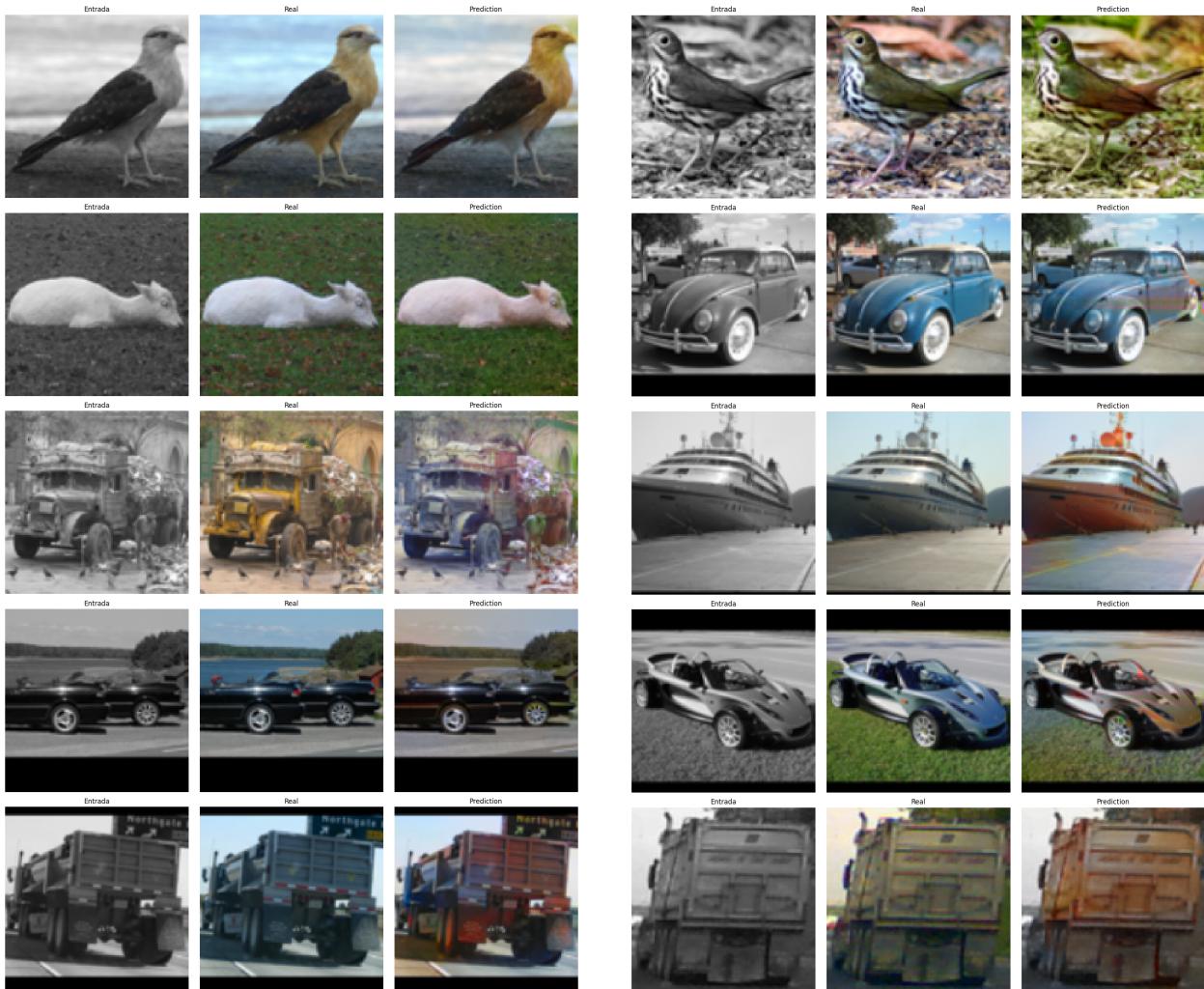


Figura 6: Resultados cualitativos de la GAN.

Como podemos ver con las GAN, las imágenes resultantes tienen colores más saturados y con mejor continuidad que con los autoencoders y VAEs, especialmente si son objetos con bordes bien definidos o texturas sencillas.

Aún así el modelo sigue mostrando ciertas ambigüedades en regiones donde el color real no está bien determinado, lo cual es esperable en coloración.

## GAN Guiada por Segmentación

Tras comprobar que la GAN clásica si que colorea las imágenes decentemente, para solucionar algunas de las ambigüedades que ya hemos mencionado, probamos a mejorar este enfoque incorporandole información semántica mediante segmentación. La motivación principal es que muchos errores cromáticos pasan por la falta de contexto, es decir, la GAN no sabe si una región gris corresponde a cielo, agua, vegetación o a un objeto en concreto.

Para resolver esto hemos complementado nuestra GAN con segmentación utilizando SegFormer, que nos proporciona una máscara semántica de la imagen pixel por pixel. Este mapa nos sirve como entrada extra para el generador, y le ayuda a colorear de forma más coherente con los diferentes elementos de la escena.

## Obtención de Máscaras Semánticas con SegFormer

Para obtener las máscaras hemos utilizado SegFormer-B0 [10], un modelo *transformer-based* pre-entrenado sobre ADE20K, que destaca por su buen equilibrio entre precisión y velocidad. Antes de escoger SegFormer como modelo de segmentación, probamos a utilizar *Segment Anything Model* (SAM) [11], dado que es uno de los métodos más recientes y precisos. Sin embargo, en la práctica SAM resultó muy costoso computacionalmente para los recursos que teníamos. Por estas razones descartamos SAM y optamos por SegFormer-B0 que es un modelo mucho más eficiente computacionalmente. Para cada imagen en RGB del dataset generamos un mapa de logits de dimensión  $(150, H, W)$ , que corresponde a las clases originales de ADE20K. Este mapa se interpola a la resolución original y se obtiene la clase más probable en cada píxel:

$$\text{mask}_{150}(x, y) = \arg \max_c \text{logits}[c, x, y].$$

Dado que 150 clases son demasiado detalladas para nuestro problema y generan ruido innecesario, las reducimos a un conjunto de 9 clases semánticas más relevantes en nuestro dataset:

- cielo,
- vegetación,
- agua,
- animales,
- persona/cuerpo,
- vehículos,
- estructuras/edificios,
- objetos pequeños,
- fondo.

Después convertimos estas etiquetas a one-hot:

$$S \in \mathbb{R}^{9 \times 128 \times 128}.$$

Estas máscaras una vez generadas las almacenamos como archivos .npy, para entrenar con ellas posteriormente.

## Integración de la Máscara en la GAN

El generador ahora recibe como entrada no solo el canal L, sino también los 9 canales de la segmentación:

$$G([L, S]) \rightarrow \hat{ab}.$$

Esto expande la entrada de la U-Net de 1 a 10 canales, haciendo que desde el primer nivel del encoder el modelo pueda ver el contenido semántico de la imagen.

El discriminador se mantiene con la misma estructura PatchGAN anterior, evaluando el par  $(L, \hat{ab})$ , ya que su función es juzgar el realismo del color, no la semántica.

## Función de Pérdida

La GAN guiada por segmentación mantiene exactamente la misma configuración de pérdidas que la GAN clásica, ya que el objetivo del modelo no cambia: generar unos canales *ab* cromáticamente coherentes con la imagen real. La introducción de la máscara semántica afecta únicamente a la entrada del generador, pero no altera el resto de la arquitectura adversarial.

Así, la pérdida total del generador combina tres términos:

$$\mathcal{L}_G = \mathcal{L}_{GAN} + \lambda_1 \mathcal{L}_{L1} + \lambda_2 \mathcal{L}_{FM},$$

donde:

- **$\mathcal{L}_{GAN}$ : pérdida adversaria.** Incentiva que el generador produzca imágenes que el discriminador clasifique como reales.
- **$\mathcal{L}_{L1}$ : pérdida de reconstrucción.** Mide la diferencia absoluta promedio entre los canales reales y generados. Su papel es estabilizar el aprendizaje y evitar desviaciones cromáticas excesivas.
- **$\mathcal{L}_{FM}$ : Feature Matching Loss.** Calcula la distancia entre las activaciones internas del discriminador para imágenes reales y generadas. Actúa como regularizador y ayuda a estabilizar el entrenamiento adversarial.

Siguiendo la misma configuración que en la GAN sin segmentación, empleamos  $\lambda_1 = 10$  y  $\lambda_2 = 10$ , que mantienen un equilibrio entre fidelidad cromática y calidad perceptual.

El discriminador utiliza la misma pérdida adversarial que en el modelo clásico:

$$\mathcal{L}_D = \mathcal{L}_{real} + \mathcal{L}_{fake},$$

evaluando únicamente si el par  $(L, \hat{ab})$  es realista, ya que su función no requiere información semántica.

## Entrenamiento

Mantenemos la misma configuración que en la GAN básica:

- Optimizador Adam ( $\text{lr} = 2 \cdot 10^{-4}$ ,  $\beta_1 = 0,5$ ),
- Imagen de entrada: canal L normalizado en  $[-1, 1]$ ,
- Máscara semántica: mapa one-hot de 9 canales,
- Resolución final:  $128 \times 128$ ,
- Batch size: 32,
- Epochs: 50.

## Resultados y Métricas

Al evaluar el rendimiento del modelo guiado con segmentación, vemos que las métricas cuantitativas son muy similares a las obtenidas por la GAN clásica. Esto es esperable, dado que las métricas como SSIM y PSNR no capturan directamente la coherencia semántica del color, sino aspectos puramente estructurales o de reconstrucción.

Cuadro 4: Métricas de evaluación de la GAN guiada por segmentación

| Métrica | GAN + Segmentación |
|---------|--------------------|
| SSIM    | 0.9073             |
| PSNR    | 23.3946            |
| LPIPS   | 0.1660             |

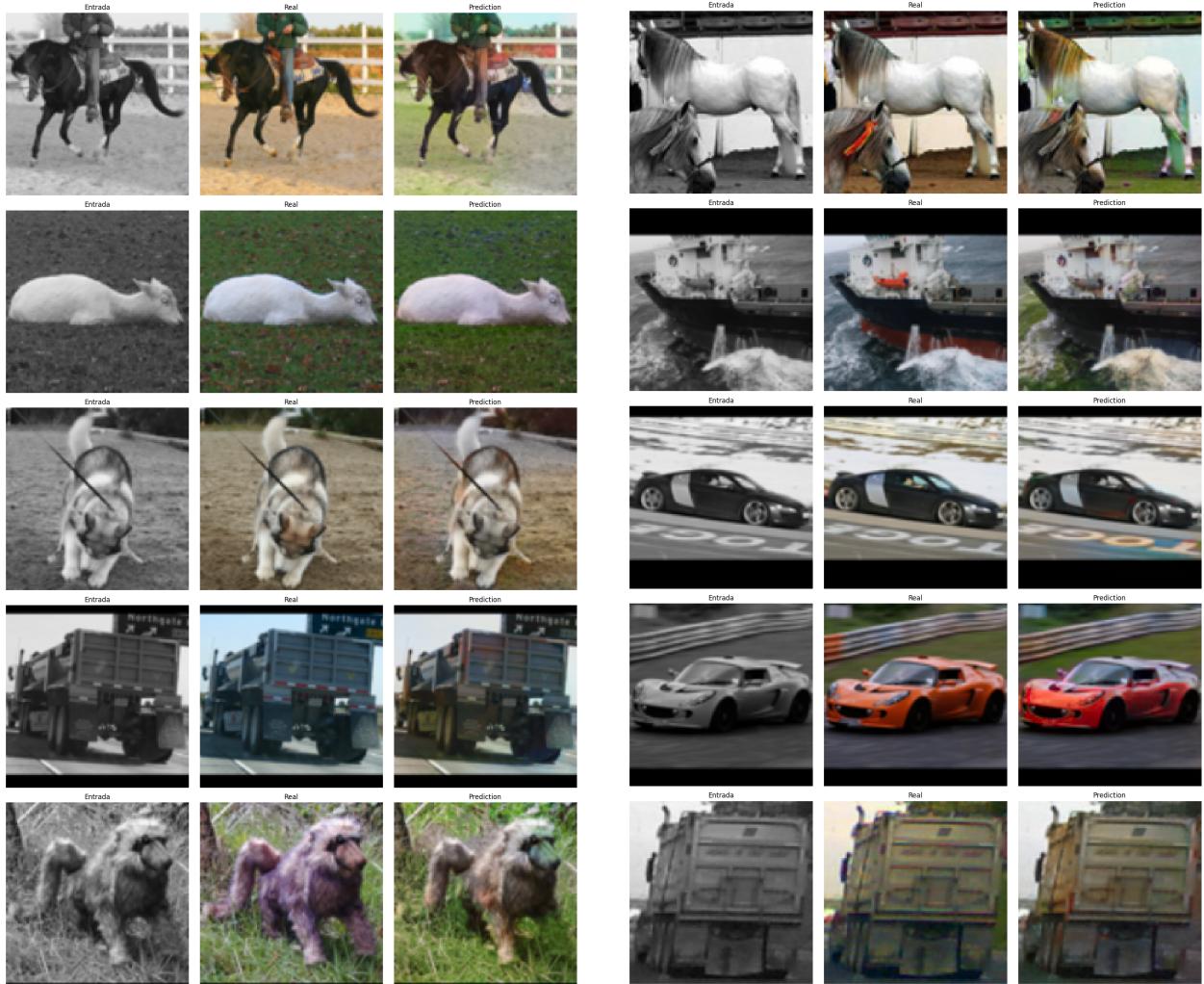


Figura 7: Resultados cualitativos del modelo con segmentación.

A nivel cualitativo, destacamos los siguientes efectos positivos derivados de la segmentación:

- **Mayor coherencia cromática en clases** como en el cielo, agua o vegetación.
- **Reducción del *color bleeding*** en contornos de objetos, especialmente en personas, animales o vehículos. La máscara impide que el color se extienda a regiones incorrectas.

Aunque las métricas muestran un rendimiento similar al de la GAN sin segmentación, el análisis visual revela mejoras en la coherencia del color. En los ejemplos del conjunto de test, se aprecia cómo la segmentación guía al generador a producir coloraciones más consistentes con el contenido semántico de la imagen.

No obstante, también observamos que cuando la segmentación de SegFormer es incorrecta, el generador puede reproducir errores similares a los de la GAN clásica o incluso colorear regiones de un color que no les corresponde, por ejemplo, colorear la tierra como si fuera césped. Aun así,

el modelo se mantiene estable y no empeora significativamente, lo que indica que la arquitectura no se degrada incluso teniendo ruido semántico.

En conjunto, aunque las métricas cuantitativas no mejoran de forma sustancial, la calidad perceptual sí lo hace ligeramente, especialmente en escenas donde la semántica es determinante para la elección del color. Cabe destacar que una de las posibles razones por las que la incorporación de segmentación no produce una mejora significativa es la baja resolución de las imágenes del dataset. Al trabajar con imágenes de  $128 \times 128$  píxeles, las máscaras generadas tienden a ser poco precisas en regiones pequeñas, lo que hace que el generador no reciba información semántica relevante. En estos casos, la segmentación no introduce un contexto semántico suficientemente rico como para influir de manera notable en el proceso de coloración.

## Coloración con modelos de difusión

Los modelos de difusión se han consolidado como una de las técnicas más potentes en generación de imágenes. Su principio básico consiste en transformar ruido gaussiano en datos estructurados mediante un proceso iterativo de denoising. Formalmente, el proceso directo añade ruido a una imagen  $x_0$  siguiendo una cadena de Markov:

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t} x_{t-1}, \beta_t I),$$

donde  $\beta_t$  controla la cantidad de ruido en el paso  $t$ . El modelo aprende la distribución inversa  $p_\theta(x_{t-1} | x_t)$  para recuperar la señal original. Tras suficientes iteraciones, se obtiene una imagen coherente a partir de ruido inicial. Estos métodos han demostrado gran capacidad en tareas de síntesis y edición de imágenes [12]. En el contexto de la coloración, los modelos de difusión resultan útiles porque permiten generar detalles cromáticos plausibles a partir de información parcial (imágenes en escala de grises), preservando bordes y texturas.

## Arquitectura

El modelo se basa en el VAE de *Stable Diffusion 1.5* como codificador–decodificador, sobre el cual se han probado distintas variantes de la arquitectura UNet:

### Entrenamiento desde cero del UNet

En este enfoque se diseñó y entrenó un UNet desde cero, inicializando todos los parámetros de manera aleatoria. La arquitectura utilizada incluye tres bloques descendentes y tres ascendentes, con mecanismos de atención en el nivel intermedio. Todos los parámetros del modelo fueron entrenados, lo que permite un control completo sobre el proceso de aprendizaje. Este modelo, pese a ser más pequeño y por lo tanto con menor capacidad, aprende directamente la tarea de coloración, especializándose desde el inicio.

### Fine-tuning del UNet de Stable Diffusion

En este caso se partió del UNet preentrenado de *Stable Diffusion*, restringiendo el ajuste únicamente al tercer bloque ascendente y al bloque intermedio. El resto de parámetros se mantuvieron

congelados, de modo que sólo estas capas fueron entrenadas. Esta estrategia reduce el coste computacional y el riesgo de sobreajuste, aprovechando el conocimiento previo del modelo y adaptándolo específicamente a la tarea de coloración. Por otro lado, esta estrategia es la más costosa computacionalmente.

### Adaptación mediante LoRA

La tercera estrategia consistió en aplicar LoRA (*Low-Rank Adaptation*) sobre el UNet de *Stable Diffusion*. Se insertaron matrices de bajo rango en las capas de atención, concretamente en los módulos de consulta y valor, con un rango de  $r = 4$ , un factor de escalado de  $\alpha = 16$  y un dropout de 0.05 para regularización. Este enfoque permite adaptar el modelo de manera eficiente, reduciendo drásticamente el número de parámetros entrenables sin necesidad de modificar el resto de la arquitectura, lo que lo convierte en una opción muy mucho más eficiente que el finetuning.

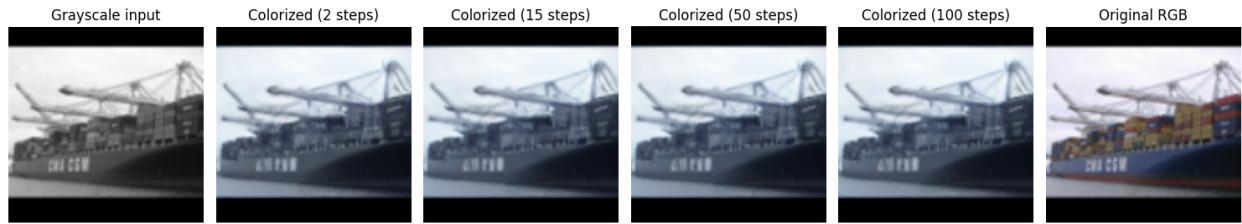
### Entrenamiento

El entrenamiento se realizó con 25 000 imágenes del dataset STL-10, que al tratarse de imágenes pequeñas ( $96 \times 96$  píxeles) limitan la capacidad del modelo para aprender detalles finos y pueden afectar la nitidez cromática. Para reducir el elevado coste computacional, se calcularon previamente las representaciones latentes con el VAE, entrenando directamente sobre ellas mediante el optimizador Adam y utilizando como función de pérdida el MSE.

En cuanto a las estrategias de adaptación, el *fine-tuning* del UNet se restringió al último bloque con el fin de disminuir el coste y evitar el sobreajuste, mientras que la variante LoRA permitió ajustar el modelo de manera eficiente con un menor número de parámetros entrenables. Dado que el proceso de entrenamiento resultó muy lento, el número de épocas fue limitado: se entrenaron 200 épocas para el UNet desde cero, 30 épocas para la variante LoRA y 20 épocas para el *fine-tuning*.

## Coloración de imágenes

Figura 8: Epoch 175 de entrenamiento del la UNet desde cero:



(a) Color scale = 1.2



(b) Color scale = 1.4



(c) Color scale = 1.6



(d) Color scale = 1.8

## Hiperparámetros

En este modelo nos encontramos 2 hiperparámetros importantes para la calidad al generar una imagen.

El primero es los **steps**, es decir, el número de pasos que se dan de eliminación de ruido. Se ha probado con un amplio rango de valores (2, 15, 50 y 100). En *Diffusing Colors: Image Colorization with Text Guided Diffusion* [12] se utiliza 50 pasos para generar. En nuestro experimento se puede ver que a partir de los 15 pasos, ya no hay diferencia.

El segundo es el **color scale**. Este hiperparámetro modifica cuanto cambia el color. Cuanto más grande el valor, más intenso será el color de la imagen. Ha resultado ser más importante el color scale en el resultado que el número de pasos. En nuestros experimentos, se ha definido 1.5 como el mejor color scale.



Figura 9: Comparación hiperparámetros en imagen de coche



Figura 10: Comparación hiperparámetros en imagen de un perro

## Resultados

Para la evaluación cuantitativa de la coloración de imágenes se han considerado tres métricas complementarias. La métrica LPIPS (*Learned Perceptual Image Patch Similarity*) se apoya en redes neuronales preentrenadas para estimar la similitud perceptual entre la imagen generada y la referencia, capturando diferencias relevantes para la percepción humana más allá de la comparación directa de píxeles. La MSE (*Mean Squared Error*) calcula el error cuadrático medio entre ambas imágenes, proporcionando una medida objetiva de la fidelidad numérica de la reconstrucción, aunque puede ser menos sensible a la percepción visual. Finalmente, la métrica CIEDE2000, propuesta por la Comisión Internacional de Iluminación, cuantifica la diferencia de color teniendo en cuenta la percepción humana; en el contexto de la coloración es especialmente importante, ya que permite evaluar si los tonos generados coinciden con los colores reales de referencia incluso en regiones donde pequeñas variaciones cromáticas resultan perceptualmente significativas. En conjunto, estas métricas ofrecen una visión más completa del rendimiento del modelo, equilibrando precisión numérica, calidad perceptual y fidelidad cromática.

Cuadro 5: Resultados de coloración con modelos de difusión.

| Modelo                      | LPIPS         | MSE           | CIEDE2000    |
|-----------------------------|---------------|---------------|--------------|
| UNet desde cero             | 0.1437        | <b>461.28</b> | 11.86        |
| Fine-tuning último bloque   | 0.1416        | 554.04        | 13.76        |
| LoRA sobre Stable Diffusion | <b>0.1322</b> | 499.08        | <b>11.77</b> |

Al analizar los resultados, se observa que los tres modelos mantienen la estructura global de las imágenes sin deformaciones, como refleja el bajo valor de LPIPS. No obstante, en la tarea de coloración se evidencian limitaciones, ya que los colores generados difieren de los tonos de referencia. La métrica CIEDE2000 permite cuantificar estas diferencias cromáticas desde una perspectiva

perceptual: valores superiores a 3 ya se consideran inaceptables en aplicaciones de control de calidad, mientras que diferencias por encima de 10 indican una discrepancia claramente notable para el observador humano [13]. En las imágenes obtenidas se aprecia que, aunque los modelos logran colorear las regiones principales, tienden a equivocarse en el tono o a producir colores más apagados de lo esperado. Entre las estrategias evaluadas, el *fine-tuning* del último bloque ofrece el peor desempeño, mientras que la variante LoRA consigue los mejores resultados perceptuales.

Los resultados muestran varias limitaciones claras:

- El modelo tiende a difuminar el texto presente en las imágenes.
- Las coloraciones presentan regiones con aspecto grisáceo o, en ocasiones, fogonazos de color incoherentes.
- El rendimiento global no alcanza niveles excelentes, aunque se observan diferencias entre las variantes de UNet.
- El número de pasos de difusión no resulta determinante en la calidad final.

No se ha logrado un modelo de difusión capaz de colorizar imágenes a un buen nivel. Los modelos del artículo Diffusing Colors [12] logran mejores resultados porque se apoyan en modelos de difusión previamente entrenados a gran escala, integran prompts textuales para guiar la coloración y aprovechan datasets masivos y diversos.

Viendo estas limitaciones, vamos a intentar atacar la única que podemos intentar resolver, integran prompts textuales para guiar la coloración.

## Coloración Guiada por Texto

El objetivo es desarrollar un modelo capaz de colorear imágenes utilizando como guía una descripción textual. La idea central es que la red no solo aprenda a reconstruir colores, sino que además incorpore información semántica explícita presente en el texto (por ejemplo: “pétalos rojos”, “centro amarillo”, etc.).

Inicialmente se intentó utilizar el dataset original con animales y vehículos del resto de apartados del proyecto. Sin embargo, dicho conjunto de datos no incluía descripciones textuales ni etiquetas, lo que dificultaba enormemente la incorporación de información lingüística. Se intentaron dos estrategias para suplir esta ausencia:

### 1. Generar descripciones automáticamente mediante un captioner preentrenado.

Resultó poco útil porque al trabajar con imágenes en escala de grises, el captioner no disponía de pistas sobre los colores reales y la información que producía era demasiado general, no aportaba señales relevantes al proceso de coloración que no entendiera ya el propio AE colorizador ya que no mencionaba nada sobre colores. Además, era más costoso de ejecutar por que requería modelo pesado durante la preparación del dataset para generar los textos, ralentizando el flujo de trabajo.

## 2. Clasificar cada imagen según su color dominante.

Este método tampoco funcionó: muchas imágenes contenían varios colores relevantes, o el color dominante procedía del fondo en vez del objeto de interés. Esto generaba etiquetas incorrectas además de prompts muy simples (nombres de los colores mayoritarios exclusivamente) que luego en la inferencia no serían iguales y hacía que el modelo no mejorase respecto al autoencoder base.

Tras comprobar que ninguno de estos dos enfoques producía beneficios reales, decidimos recurrir a un dataset ya anotado con descripciones ricas y con referencias explícitas a colores.

## Dataset empleado

Seleccionamos el dataset `Oxford Flowers102`, que incluye para cada imagen de la flor correspondiente, una descripción textual detallada. Estas descripciones suelen mencionar explícitamente los colores de los pétalos, hojas o centros, lo cual resulta ideal para un modelo de coloración guiada.

Para asegurar que el texto proporcionado realmente aportaba información cromática, construimos una lista de palabras relacionadas con colores (alrededor de 40 términos, incluyendo principalmente nombres de colores y tonalidades) y solo se conservaron aquellas imágenes cuyo texto contenía al menos una palabra de la lista. Con esto garantizamos que el modelo recibe mensajes relevantes sobre el color que queremos obtener.

## Arquitectura del modelo

El modelo desarrollado se basa en un **UNet** con condicionamiento textual mediante capas FiLM. Se trata de un diseño autoencoder con skip-connections, que permite preservar detalles espaciales mientras el texto influye en la reconstrucción del color.

### Embedding textual

Para transformar las descripciones en vectores utilizables por la red, empleamos un **modelo CLIP preentrenado** (`openai/clip-vit-base-patch32`). Únicamente se utilizó su codificador textual para obtener un embedding robusto, compacto y semánticamente rico del texto que luego pasar al AE colorizador. Además, al ser un modelo transformer grande, el embedding representa no solo palabras de color, sino también la estructura semántica del texto, proporcionando al modelo una guía contextual (por ejemplo: si el color se refiere a los pétalos, al centro, a las hojas, etc.).

### Estructura general

- **Encoder:** cuatro bloques convolucionales que reducen progresivamente la resolución y extraen características visuales de nivel creciente.
- **Bottleneck:** un bloque central donde se combina la información visual comprimida con la información textual y se enriquece.
- **Decoder:** cuatro bloques de deconvolución que reconstruyen la imagen a color, fusionando la información procesada con los mapas del encoder mediante conexiones residuales. El

último bloque del decoder produce la imagen RGB resultante (salida en 3 canales).

### Bloques convolucionales

Cada bloque convolucional del modelo está compuesto por dos operaciones principales. Una capa convolucional, que permiten extraer y transformar características visuales de forma progresiva. Y una capa de normalización por batch, aplicada tras las convoluciones para estabilizar la distribución de activaciones y facilitar el entrenamiento.

A partir de esta estructura común, el modelo utiliza dos variantes según la etapa. En el encoder, los bloques emplean convoluciones estándar que reducen la resolución espacial de la imagen, capturando patrones cada vez más globales. En el decoder, los bloques recurren a convoluciones transpuestas, que aumentan la resolución y permiten reconstruir la imagen final a partir de las representaciones comprimidas.

### Condicionamiento textual

El componente clave del modelo son las capas **FiLM** (Feature-wise Linear Modulation). Estas capas permiten que el texto modifique directamente la activación de las capas convolucionales mediante dos parámetros aprendidos:  $\gamma$  que controla la amplificación o atenuación por canal, y  $\beta$  que desplaza la activación por canal.

La operación es:

$$\text{FiLM}(X) = X \cdot (1 + \gamma) + \beta$$

donde cada canal de la activación se ajusta en función del embedding textual. logrando que si el texto menciona “petals are bright red”, las FiLM potencian las características asociadas a tonos rojizos en regiones que visualmente parezcan pétalos; si describe “dark green leaves”, favorece la aparición de verdes más apagados en la zona de hojas.

El condicionamiento se aplica por primera vez en el bottleneck, donde se controla la representación más comprimida de la imagen, y luego se continua aplicando en las etapas del decoder, donde se ajusta el color reconstruido de forma más localizada según los FiLM, que actúan integrando la semántica lingüística en el flujo visual del UNet.

### Entrenamiento

Durante el proceso de entrenamiento se utilizó una función personalizada `collate_fn` para garantizar que cada imagen queda alineada con su descripción textual antes de introducirse en el modelo.

Para la etapa de aprendizaje se empleó **MSE (Mean Squared Error)** como función de pérdida, comparando píxel a píxel la imagen generada con la imagen real. Para tarea de coloración resulta ser de las más útiles ya que consiste en aproximar colores continuos. El modelo se entrenó sobre la partición de entrenamiento y se validó periódicamente para monitorizar su rendimiento.

## Resultados

Para evaluar el desempeño del modelo se definió un procedimiento de testeo que carga el modelo entrenado y la sección de los datos reservada para pruebas. Los resultados se presentan en forma de tríos de imágenes: (imagen en gris, predicción generada, referencia real) acompañados del prompt textual.

Los resultados muestran que el modelo es capaz de producir una coloración muy buena por regiones salvo en algunos casos concretos donde confunde los pétalos con hojas. Suponemos que debido a que las descripciones textuales contienen detalles sobre colores y partes de la flor, se logra que el modelo aplique los colores en las zonas adecuadas gracias a que la información del texto complementa la estructura visual.

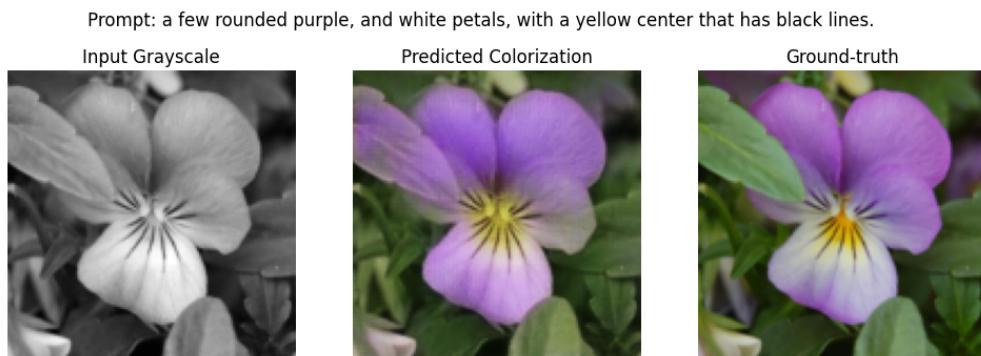


Figura 11: Ejemplo de resultado de coloración guiada por texto (1).



Figura 12: Ejemplo de resultado de coloración guiada por texto (2).

Prompt: three layers of thick green petals surrounding thousands of vibrant purple stamen.

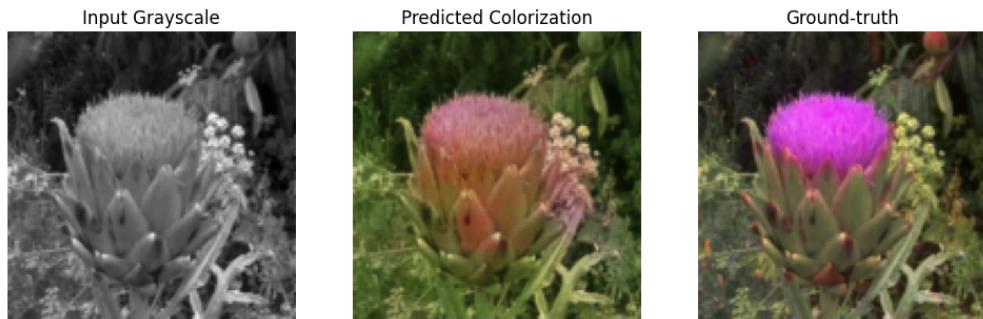


Figura 13: Ejemplo de resultado de coloración guiada por texto (3).

Prompt: this flower has long upturned petals with ruffles edges which are yellow with red centers.

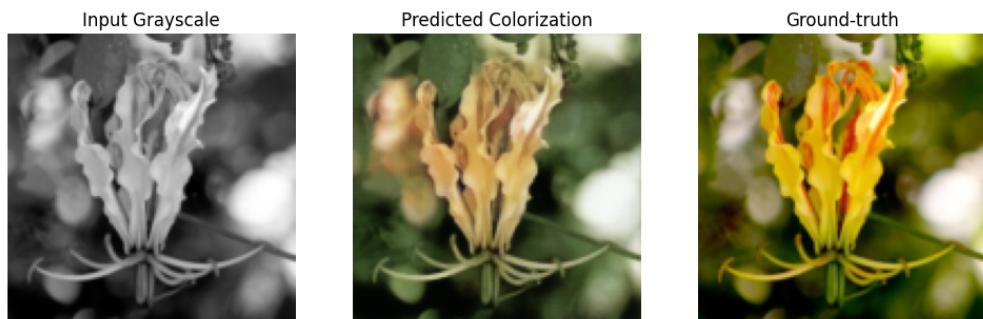


Figura 14: Ejemplo de resultado de coloración guiada por texto (4).

## Conclusiones

En esta sección se presentan las conclusiones generales del trabajo, apoyadas en las métricas cuantitativas y cualitativas obtenidas para cada modelo de coloración. Los resultados permiten comparar de manera objetiva el rendimiento de arquitecturas basadas en autoencoders, variational autoencoders, redes generativas adversarias y modelos de difusión, destacando sus fortalezas y limitaciones en términos de estructura, fidelidad numérica, percepción visual y fidelidad cromática.

Cuadro 6: Comparación de métricas entre modelos de coloración

| Modelo                | SSIM ↑        | LPIPS ↓       | MSE ↓        | CIEDE2000 ↓ |
|-----------------------|---------------|---------------|--------------|-------------|
| U-Net (AE)            | 0.9323        | 0.1345        | <b>190.6</b> | <b>8.76</b> |
| ResU-Net (AE)         | <b>0.9360</b> | 0.1466        | 203.7        | 8.89        |
| VAE Denso             | 0.0600        | 0.6884        | 15103.6      | –           |
| VAE Convolutional     | 0.9338        | 0.1485        | 200.5        | 8.84        |
| GAN clásica           | 0.9097        | 0.1628        | 278.7        | –           |
| GAN + Segmentación    | 0.9073        | 0.1660        | 297.9        | –           |
| Diffusion UNet        | 0.9152        | 0.1437        | 461.3        | 11.86       |
| Diffusion Fine-tuning | 0.8636        | 0.1416        | 554.0        | 13.76       |
| LoRA Stable Diffusion | 0.9034        | <b>0.1322</b> | 499.1        | 11.77       |

El análisis de la Tabla 6 muestra que los autoencoders (U-Net y ResU-Net) son los más consistentes en métricas objetivas: presentan la mayor coherencia estructural (SSIM), los menores errores numéricos (MSE) y, crucialmente, la mejor fidelidad cromática ( $\text{CIEDE2000} \approx 8.8$ ), superando tanto a GAN como a modelos de difusión. Entre los VAE, el enfoque convolucional se aproxima a los AE en SSIM/MSE y también mantiene un CIEDE2000 bajo, mientras que el VAE denso queda descartado por su pobre desempeño.

Las GAN ofrecen mayor saturación y continuidad de color a nivel visual, aunque sus métricas cuantitativas son inferiores a las de los AE y no disponen de CIEDE2000 en esta evaluación. Los modelos de difusión, pese a mejorar la similitud perceptual según LPIPS (especialmente con LoRA), muestran peores diferencias de color perceptuales (CIEDE2000 entre 11.8 y 13.8), indicando que aún no igualan la fidelidad cromática lograda por los autoencoders.

En conjunto, los resultados señalan que los AE son el referente en estructura, fidelidad numérica y color (CIEDE2000), las GAN mejoran la naturalidad cromática de forma cualitativa, y los modelos de difusión destacan en LPIPS pero requieren integrar más contexto (semántico/textual) para cerrar la brecha en color. Es decir, requieren más entrenamiento.

## Coloración guiada por texto

La coloración guiada por texto ha demostrado ofrecer resultados muy sólidos cuando se dispone de imágenes acompañadas de descripciones ricas y precisas, como ocurre en el caso de las flores. Consideramos que esta estrategia podría extenderse a otros dominios donde la coloración sea relevante; sin embargo, la principal limitación reside en la disponibilidad de *captions* de calidad, algo poco común en la mayoría de los datasets existentes. Pensamos que sería factible aproximarse a un colorizador más general para todos los usos si se adoptara un enfoque de entrenamiento similar al de modelos como CLIP, que aprovechan casi todas las imágenes que hay en Internet junto a su texto alternativo. No obstante, un esfuerzo de esta magnitud requeriría modelos de mayor capacidad y recursos computacionales sustancialmente superiores.

## Referencias

- [1] Xiaoyang Kang et al. “DDColor: towards photo-realistic image colorization via dual decoders”. En: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. de 2023, págs. 328-338.
- [2] Zhexin Liang et al. “Control color: multimodal diffusion-based interactive image colorization”. En: *arXiv preprint arXiv:2402.10855* (2024). URL: <https://arxiv.org/abs/2402.10855>.
- [3] Maria Larchenko et al. “Color transfer with modulated flows”. En: *arXiv preprint arXiv:2503.19062* (2025). URL: <https://arxiv.org/abs/2503.19062>.
- [4] Adam Coates, Andrew Ng y Honglak Lee. “An analysis of single-layer networks in unsupervised feature learning”. En: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*. 2011, págs. 215-223.
- [5] Olaf Ronneberger, Philipp Fischer y Thomas Brox. “U-Net: convolutional networks for biomedical image segmentation”. En: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Springer, 2015, págs. 234-241.
- [6] Zizhao Zhang, Qi Liu y Yefeng Wang. “Road extraction by deep residual U-Net”. En: *IEEE Geoscience and Remote Sensing Letters* 15.5 (2018), págs. 749-753.
- [7] Diederik P. Kingma y Max Welling. “Auto-encoding variational Bayes”. En: *arXiv preprint arXiv:1312.6114* (2013). URL: <https://arxiv.org/abs/1312.6114>.
- [8] Phillip Isola et al. “Image-to-image translation with conditional adversarial networks”. En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.
- [9] Tim Salimans et al. “Improved techniques for training GANs”. En: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016.
- [10] Enze Xie et al. “SegFormer: simple and efficient design for semantic segmentation with transformers”. En: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.
- [11] Alexander Kirillov, Eric Mintun, Nikhila Ravi et al. “Segment anything”. En: *arXiv preprint arXiv:2304.02643* (2023). URL: <https://arxiv.org/abs/2304.02643>.
- [12] Nir Zabari et al. “Diffusing colors: image colorization with text guided diffusion”. En: *arXiv preprint arXiv:2312.04145* (2023). DOI: 10.48550/arXiv.2312.04145. URL: <https://arxiv.org/abs/2312.04145>.
- [13] M. R. Luo, G. Cui y B. Rigg. “The development of the CIE 2000 colour-difference formula: CIEDE2000”. En: *Color Research & Application* 26.5 (2001), págs. 340-350. DOI: 10.1002/col.1049. URL: <https://doi.org/10.1002/col.1049>.