UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

INSTITUTO DE INVESTIGACIÓN EN CIENCIAS BÁSICAS Y APLICADAS

CENTRO DE INVESTIGACIÓN EN INGENIERÍA Y CIENCIAS APLICADAS

DEEP LEARNING APLICADO A INPAINTING EN RETRATOS HABLADOS

TESIS PARA OBTENER EL GRADO DE MAESTRÍA EN INGENIERÍA Y CIENCIAS APLICADAS

Presenta

JOSÉ MARCELO FRÍAS CASTRO

DIRECTOR(ES): DR. ROBERTO ALAN BELTRÁN VARGAS

CODIRECTOR: DR. BRAULIO MISAEL VILLEGAS MARTÍNEZ

CUERNAVACA, MORELOS **NOVIEMBRE, 2026**

Instituto de Investigación en Ciencias Básicas y Aplicadas

Índice

Lista de Tablas **5**

Lista de Figuras **7**

Resumen **9**

Abstract **10**

Agradecimientos **11**

CAPÍTULO 1: ANTECEDENTES **13**
1.1 INTRODUCCIÓN 13
1.2 ESTADO DEL ARTE 15
1.3 JUSTIFICACIÓN 26
1.4 HIPÓTESIS 27
1.5 OBJETIVOS 28

CAPÍTULO 2: MARCO TEÓRICO **29**
2.1 Teoría Fundamental 29
2.1.1 La Imagen Digital 29
2.1.2 Visión Artificial 30
2.1.3 Procesamiento Digital de Imágenes (PDI) 30
2.2 Inpainting 31
2.2.1 Técnicas Tradicionales de Inpainting. 33
2.2.2 Métodos basados en parches. 33
2.2.3 Métodos basados en difusión. 34
2.3 Inteligencia Artificial 34
2.3.1 Aprendizaje Automático. 35
2.3.2 Redes Neuronales. 36
2.3.3 Deep Learning (DL) 38
2.3.4 Redes Neuronales Convolucionales (CNN) 39
2.3.5 Funciones de transferencia 41

2.3.5.1 Tangente Hiperbólica 41
2.3.5.2 ReLU 42
2.3.5.3 Leaky ReLU 43
2.3.5.4 Parametric ReLU 43
2.3.6 Redes Generativas Adversariales (GAN) 44
2.4 Aumento de Datos 46
2.5 Autocodificadores 48
2.6 Autocodificadores de eliminación de ruido 50
2.7 Arquitectura U-Net 51
2.8 Inpainting ciego y no ciego 53
2.9 Métricas y funciones de pérdida para inpainting 55
2.9.1 Error L1: 55
2.9.2 Error L2: 56
2.9.3 Proporción máxima de señal a ruido (PSNR): 56
2.9.4 Structural Similarity Index Measure (SSIM): 58
2.9.5 Distancia de Fréchet de Inception (FID): 61
2.9.6 Similitud perceptual aprendida de parches de imagen (Learned Perceptual Image Patch Similarity, LPIPS): . 62
2.9.7 Funciones de pérdida para inpainting 64

CAPÍTULO 3: DESARROLLO TEÓRICO EXPERIMENTAL, RESULTADOS Y DISCUSIÓN 66
3.1 Equipo 66
3.1.1 Estación de trabajo 1 66
3.1.2 Estación de trabajo 2 66
3.2 Materiales: lenguaje de programación, librerías 67
3.2.1 Grupos Funcionales Principales 70
3.2.2 Dependencias Excluidas de la Tabla Principal 71
3.3 Metodología experimental y diseño iterativo 72
3.3.1 Creación del conjunto de datos para entrenamiento . 72
3.3.2 Creación de un flujo de trabajo de entrenamiento . . . 75

3.3.3 Diseño y resultados de la iteración de entrenamiento de un autocodificador de eliminación de ruido 82
3.3.3.1 Conclusiones de la primera iteración de entrenamiento 110
3.4 Evaluación cuantitativa, ajuste fino y discusión de resultados 112
3.4.1 Desafíos principales del análisis cuantitativo de resultados 114
3.4.2 Objetivos específicos del sistema de evaluación . . . 115
3.4.3 Arquitectura del sistema de evaluación 117
3.4.4 Complementariedad de Métricas 122
3.4.5 Resultados de métricas de calidad de imagen y ajuste fino 124
3.4.6 Comparación de métricas de calidad de imagen con el estado del arte 138
3.4.7 Conclusiones del análisis con métricas de calidad de imagen y ajuste fino 140

CONCLUSIONES **146**
RECOMENDACIONES 150

REFERENCIAS **152**

APÉNDICES **160**
Apéndice 1: Archivo environment.yml 160

ANEXOS **164**
Anexo 1: Participación en el Congreso Nacional de Física 164
Anexo 2: Participación en taller o curso para la retribución social . 166

Lista de Tablas

1 Dependencias principales utilizadas en el flujo de trabajo de reconstrucción de imágenes con CNN. 67
2 Resultados cuantitativos del barrido de probabilidad de Dropout. 86
3 Resultados cuantitativos del barrido de tasa de aprendizaje. . 87
4 Resultados cuantitativos del barrido del planificador de tasa de aprendizaje. 88
5 Resultados cuantitativos del barrido de optimizadores. 89
6 Resultados cuantitativos del barrido de parámetros β_1 y β_2. . 90
7 Resultados cuantitativos del barrido de inicializadores de pesos 91
8 Resultados cuantitativos del barrido de normalización. 93
9 Resultados cuantitativos del barrido de tipos de máscara. . . 95
10 Resultados cuantitativos de las funciones de pérdida. 96
11 Resultados cuantitativos del barrido de interpolación y conexiones. 98
12 Resultados cuantitativos de la comparación entre distintos tamaños de cuello de botella. 100
13 Resultados cuantitativos del barrido de tamaño de profundidad de la red. 102
14 Resultados cuantitativos del barrido de funciones de activación.105
15 Resultados cuantitativos de la comparación entre modalidades de inpainting ciego y no ciego. 107
16 Resultados cuantitativos de barrido de tamaño de lote. 109
17 Complementariedad de las métricas de calidad de imagen calculadas. 123
18 Estadísticas de evaluación del modelo best_model_NONBLIND_V26_UNET_PRELU6B_MIX_LOSS (10 ejecuciones). 128

19 Estadísticas de evaluación del modelo best_model_NONBLIND_V27_UNET_PRELU7B_MIX_LOSS (10 ejecuciones). 130
20 Estadísticas de evaluación del modelo best_model_NONBLIND_V29_UNET_NON_SHARED_PRELU7B_MIX_LOSS (10 ejecuciones). 132
21 Estadísticas de evaluación del modelo best_model_NONBLIND_V28_UNET_NON_SHARED_PRELU6B_MIX_LOSS (10 ejecuciones). 134
22 Resultados LPIPS del modelo best_model_NONBLIND_V28_UNET_NON_SHARED_PRELU6B_MIX_LOSS calculados con AlexNet sobre el 25% del conjunto de validación. . 135
23 Comparación final de métricas de calidad de imagen de las versiones V26, V27, V28 y V29 (media $\pm$ desviación estándar). Los encabezados indican el tamaño de entrada descrito para cada versión. 137
24 Referencia contextual y comparación aproximada de métricas con el estado del arte. 139
25 Estadísticas de evaluación del modelo v26_WITH_256_INPUT_SIZE (1 ejecución). 142
26 Estadísticas de evaluación del modelo v26_WITH_128_INPUT_SIZE (1 ejecución). 143

Lista de Figuras

1 De izquierda a derecha: Imagen original, máscara MCAR, máscara Rectangle Generator, máscara Pattern Generator, máscara Mixture. 75
2 Flujo de trabajo de entrenamiento desarrollado. 81
3 Comparación cualitativa entre inicialización de máscaras con valores RGB 127 (izquierda) y valor de la media de ImageNet (derecha). 94
4 Comparación cualitativa entre función de pérdida original (izquierda) y función de pérdida propuesta por Zhao et al., 2017 (derecha) 97
5 Comparación cualitativa entre modelo base (izquierda), conexiones de salto concatenadas (centro) y conexiones tipo U-Net (derecha) 99
6 Comparación cualitativa entre modelo base (izquierda), cuello de botella de 512 filtros (centro) y 384 filtros (derecha) 101
7 Comparación cualitativa entre modelo base (izquierda), modelo con 5 bloques convolucionales (centro) y 3 bloques convolucionales (derecha) 103
8 Comparación cualitativa entre modelo base tansig (izquierda) y modelo PReLU (derecha) 106
9 Comparación cualitativa entre modelo base (izquierda), modelo no ciego (centro) y modelo no ciego + PReLU (derecha) . 108
10 Comparación cualitativa entre modelo base con tamaño de lote 128 (izquierda), tamaño de lote 64 (centro) y tamaño de lote 32 (derecha) 110
11 Componentes principales de la arquitectura del sistema de evaluación 117
12 Resultados cualitativos de inpainting (UNET Versión 26) . . . 129
13 Resultados cualitativos de inpainting (UNET Versión 27) . . . 131
14 Resultados cualitativos de inpainting (UNET Versión 29) . . . 133

15 Resultados cualitativos de inpainting (UNET Versión 28) . . . 136
16 Resultados cualitativos de inpainting con tamaño de entrada mayor al entrenamiento (UNET Versión 26) 142
17 Resultados cualitativos de inpainting con tamaño de entrada menor al entrenamiento (UNET Versión 26) 143

Resumen

Reconociendo la necesidad de complementar un sistema de retratos hablados mediante el uso de inteligencia artificial, en el presente trabajo se desarrolla y evalúa un modelo de inpainting para la reconstrucción de rostros humanos, enfocado en rellenar secciones faltantes del rostro a partir de los rasgos faciales principales. La metodología propuesta genera de forma dinámica, mediante aumento de datos (data augmentation), un conjunto de datos basado en imágenes enmascaradas provenientes de bases de datos faciales conocidas, con el objetivo de simular escenarios en los que se dispone de los rasgos faciales principales y existen espacios vacíos entre ellos.

Esta investigación se lleva a cabo mediante un proceso experimental iterativo, en el cual se entrenan y comparan distintas configuraciones de modelos de redes neuronales convolucionales (Convolutional Neural Networks, CNNs) basados en autocodificadores de eliminación de ruido (Denoising Autoencoders) con conexiones tipo U-Net. El desempeño de los modelos se evalúa mediante métricas de calidad de imagen y observación cualitativa de las reconstrucciones generadas. Este trabajo tiene aplicaciones potenciales en seguridad y justicia, especialmente para mejorar la identificación visual en sistemas de retratos hablados.

Abstract

Recognizing the need to complement a facial composite system through the use of artificial intelligence, this work develops and evaluates an inpainting model for the reconstruction of human faces, focused on filling missing facial regions from the main facial features. The proposed methodology dynamically generates, through data augmentation, a dataset based on masked images from known facial databases, with the aim of simulating scenarios in which the main facial features are available and empty regions exist between them.

This research is carried out through an iterative experimental process, in which different configurations of convolutional neural network models based on Denoising Autoencoders with U-Net-style connections are trained and compared. The performance of the models is evaluated through image quality metrics and qualitative observation of the generated reconstructions. This work has potential applications in security and justice, particularly for improving visual identification in facial composite systems.

Agradecimientos

Agradezco a la Secretaría de Ciencia, Humanidades, Tecnología e Innovación (SECIHTI) por el apoyo otorgado mediante la beca que hizo posible la realización de mis estudios de posgrado y el desarrollo de este trabajo.

También agradezco al Centro de Investigación en Ingeniería y Ciencias Aplicadas por haber sido un espacio de formación, aprendizaje y crecimiento académico, en el que pude llevar a cabo este proyecto y adquirir conocimientos valiosos para mi formación profesional.

De manera especial, quiero agradecer a mi asesor de tesis, el Dr. Roberto Alan Beltrán Vargas, por su guía, paciencia y apoyo constante durante este proceso. Más allá de la dirección académica de este trabajo, le agradezco por haber sido una de las personas que me acercó por primera vez al mundo de la programación y por ayudarme a desarrollar habilidades y confianza que, en otro momento, no me habría atrevido siquiera a intentar. Su confianza en mi potencial y su apoyo en los momentos más difíciles fueron fundamentales no solo para la realización de esta tesis, sino también para mi crecimiento personal y profesional. Gracias por ver fortaleza en mí cuando yo mismo no siempre podía verla.

Quiero expresar también mi gratitud a mis queridos amigos y compañeros John, Kevin, Paco y Charlie. Su amistad ha sido uno de los regalos más valiosos que he tenido. Gracias a ustedes, la maestría no fue simplemente un proceso académico más; con su compañía, estos estudios de posgrado también fueron una experiencia llena de apoyo, conversaciones, risas y momentos que no voy a olvidar jamás. Estuvieron conmigo en las buenas, en las malas y en todo lo que hubo en medio. No siempre encuentro las palabras suficientes para decir lo mucho que significan para mí, pero quiero dejar claro que sin ustedes no podría haber llegado hasta aquí.

A mis padres y a mi hermano, les agradezco con todo mi corazón por su apoyo incondicional durante estos años. Gracias por creer en mis sueños incluso cuando parecían inciertos, por ayudarme a sostenerlos y por darme un hogar en el que pude sentirme seguro mientras desarrollaba mis habilidades y perseguía mis metas. Su amor, paciencia y confianza han sido una de las bases más importantes de mi vida. Mucho de lo que soy y de lo que he podido construir nace del apoyo que siempre he encontrado en ustedes. Nada de esto habría sido posible sin ustedes.

CAPÍTULO 1: ANTECEDENTES

1.1 INTRODUCCIÓN

En el contexto de complementar un sistema de retrato hablado implementado mediante sistemas lineales, se espera que exista una cantidad considerable de espacio vacío entre los rasgos faciales emitidos por dicho sistema.

La problemática consiste entonces en completar estos espacios y las regiones circundantes con datos plausibles para obtener un rostro visualmente coherente. Para este propósito se desarrolla un modelo de inteligencia artificial enfocado al inpainting, también conocido como rellenado de imágenes, buscando un balance entre la calidad de las reconstrucciones y el costo computacional.

El presente trabajo parte entonces de la implementación de un modelo de redes neuronales convolucionales basado en la arquitectura de los autocodificadores de eliminación de ruido. Se evalúa el desempeño general de esta técnica de inteligencia artificial aplicada a este problema en particular, analizando tanto las métricas cuantitativas de sus resultados como su costo computacional y el tiempo de sus procesos de entrenamiento. Si bien inicialmente se consideró la implementación de una red generativa adversarial (generative adversarial network, GAN) como una posible alternativa metodológica, el alcance del presente trabajo se delimitó al desarrollo, ajuste y evaluación de modelos convolucionales basados en autocodificadores de eliminación de ruido y conexiones tipo U-Net. Esta decisión permitió concentrar la experimentación en la comparación iterativa de distintas configuraciones de estos modelos, dejando la implementación de una GAN fuera del desarrollo experimental.

Para entrenar el modelo descrito anteriormente, se genera un conjunto de datos de forma dinámica, haciendo uso de la técnica de aumento de datos, basándose en el enmascaramiento aleatorio y la adición de ruido a imágenes de rostros humanos pertenecientes a una base de datos conocida. Esto se realiza con el fin de obtener un modelo de inpainting más robusto. Estas imágenes aumentadas son emparejadas automáticamente con los rostros completos de las imágenes originales, con el propósito de tener una referencia a partir de la cual calcular el error de reconstrucción para el modelo.

El código, el modelo y el flujo de trabajo (pipeline) de generación del conjunto de datos se encuentran adecuadamente documentados. De esta forma se da pauta a que en el futuro, otros investigadores sean capaces de replicar nuestros resultados, aplicar su propio enfoque de investigación para la resolución de este problema, o bien, utilizar el conjunto de datos para la resolución de una problemática completamente diferente.

1.2 ESTADO DEL ARTE

A continuación se presenta en orden cronológico una compilación de trabajos de investigación relevantes al ámbito del inpainting basado en redes neuronales artificiales:

Vincent et al. (2008) describe uno de los primeros trabajos en la literatura en implementar de forma práctica la arquitectura de los Autocodificadores de eliminación de ruido. En este trabajo se resalta el concepto de la corrupción de datos de entrada mediante ruido para el entrenamiento de un Autocodificador y cómo este concepto puede ser utilizado para que el modelo sea capaz de aprender representaciones robustas y características abstractas en la representación de su espacio latente. Su contribución clave consiste en demostrar cómo los Autocodificadores de eliminación de ruido podían ser utilizados como un método efectivo para técnicas de preentrenamiento en redes neuronales profundas (Deep Neural Networks, DNN), de forma que se mejorara la extracción de características y la generalización del modelo. Su implementación se llevó a cabo en tareas de clasificación de imágenes con la base de datos MNIST y fue capaz de superar el desempeño de Autocodificadores estándar.

Xie et al. (2012) propone un modelo de inpainting de imágenes basado en autocodificadores dispersos apilados de eliminación de ruido (Stacked Sparse Denoising Autoencoders, SSDAs), su idea principal consiste en combinar 2 conceptos principales: Los Autocodificadores de eliminación de ruido y los Autocodificadores Dispersos (Sparse Autoencoders). Mientras que un autocodificador de eliminación de ruido se enfoca en la reconstrucción de una entrada (input) aprendiendo a ignorar ruido e información corrompida, un autocodificador disperso busca obtener una salida (output) mediante un comportamiento disperso en las capas ocultas, es decir que para cada iteración de entrenamiento solo un subconjunto aleatorio del total de neuronas disponibles en el modelo se active, lo cual produce resultados

más robustos en el conjunto de datos de validación. Su otra aportación consiste en probar que a mayor complejidad y diversidad (realismo) tenga el ruido añadido al conjunto de entrenamiento, más eficiente será el modelo para removerlo en el conjunto de validación (menor Proporción Máxima de Señal a Ruido; Peak Signal-to-Noise Ratio, PSNR). A pesar de que estas arquitecturas ya no son utilizadas en la actualidad, es de los SSDA de donde se originan técnicas de entrenamiento para modelos de inpainting como la desactivación aleatoria de neuronas (Dropout) en modelos convolucionales modernos.

Sohn et al. (2015) introduce el concepto de autocodificador variacional condicional (Conditional Variational Autoencoder, CVAE), un Autocodificador Variacional capaz de aprender a generar una salida estructurada "y" dado ruido gaussiano y una entrada estructurada "x". Su enfoque se diferencia de los Autocodificadores Variacionales estándar en el sentido de que es capaz de generar una salida condicionada que pertenezca a una clase completamente diferente que su entrada. Esto es particularmente útil en tareas de segmentación semántica donde dada una entrada que condicionará al sistema (una imagen) se obtendrá una salida correspondiente (una máscara semántica etiquetada). Otra de las fortalezas e innovaciones de esta arquitectura es que es capaz de generar múltiples salidas posibles para una sola entrada, a diferencia de otros modelos determinísticos de Redes Neuronales Convolucionales (CNNs). Su modelo se entrenó haciendo uso del marco de trabajo (framework) Stochastic Gradient Variational Bayes (SGVB) e incorporó técnicas avanzadas de entrenamiento entre las que se incluyen objetivos de predicción multiescala e inyección de ruido a la entrada. Su modelo se aplicó a las bases de datos MNIST, Labeled Faces in the Wild y Caltech-UCSD Birds 200 y fue capaz de competir con los modelos del estado del arte de aquel año.

Pathak et al. (2016) propone los codificadores de contexto (Context Encoders), una arquitectura convolucional tipo codificador-decodificador entrenada para predecir el contenido de regiones faltantes de una imagen a

partir de la información visual circundante. Su enfoque guarda relación con los Autocodificadores de eliminación de ruido, aunque plantea una corrupción espacial de mayor tamaño y más semánticamente compleja, ya que el modelo no solo debe eliminar ruido local sino generar una hipótesis plausible para una zona faltante completa. Para lograr esto, se combina una pérdida de reconstrucción con una pérdida adversarial y se observa que el uso exclusivo de reconstrucción tiende a producir resultados borrosos, mientras que la pérdida adversarial favorece reconstrucciones visualmente más nítidas. Además, una de sus contribuciones más relevantes consiste en el análisis de diferentes estrategias de enmascaramiento de imágenes, incluyendo regiones centrales, bloques aleatorios y regiones irregulares, mostrando que las máscaras fijas pueden inducir al modelo a sobreajustarse (overfitting) a bordes o patrones de bajo nivel asociados a la máscara, mientras que las máscaras más aleatorias e irregulares favorecen la extracción de características más generales.

Zhao et al. (2017) explora el papel de las funciones de pérdida (loss functions) en las tareas de visión artificial como la eliminación de ruido (denoising), la eliminación de artefactos tipo mosaico (demosaicking) y la superresolución de imágenes (image super-resolution). Para ese punto en el estado del arte la mayoría de los modelos dedicados a resolver este tipo de problemáticas utilizaban por defecto la pérdida L2 (L2 loss, MSE). Sin embargo, a lo largo de este artículo se demuestra que funciones de pérdida alternativas, particularmente aquellas motivadas perceptualmente como lo son el índice de similitud estructural (SSIM) y su variante, el índice multiescala de similitud estructural (MS-SSIM) mejoran considerablemente la calidad de las reconstrucciones para la misma arquitectura. Asimismo, tal vez su contribución más importante fue la propuesta de una nueva función de pérdida definida como pérdida mixta (Mix Loss), la cual combina las fortalezas de la fidelidad píxel a píxel de la pérdida L1 (L1 loss, MAE) con la consistencia estructural de MS-SSIM. Esto se logra calculando la pérdida L1 mediante un mecanismo multiescala, utilizando una ventana corrediza de núcleo gaus-

siano (Gaussian kernel).

$$\mathcal{L}^{\text{Mix}} = \alpha \cdot \mathcal{L}^{\text{MS-SSIM}} + (1 - \alpha) \cdot G^{M}_{\sigma_G} \cdot \mathcal{L}^{\ell_1}$$

donde α controla el peso relativo entre la pérdida estructural y la pérdida píxel a píxel, $\mathcal{L}^{\text{MS-SSIM}}$ corresponde a la pérdida basada en MS-SSIM, $\mathcal{L}^{\ell_1}$ representa el término de pérdida L1 y $G^{M}_{\sigma_G}$ corresponde al suavizado gaussiano utilizado para ponderar dicho término en la operación de convolución.

Además de esta formulación, se destaca que la capa de pérdida había recibido poca atención frente al diseño de arquitecturas de Redes Neuronales Convolucionales, a pesar de ser el componente que guía directamente el aprendizaje del modelo. Se observa que la función de pérdida L2 y la métrica PSNR no siempre se correlacionan de manera adecuada con la percepción humana de calidad visual, mientras que L1 ayuda a preservar luminancia y color, MS-SSIM aporta consistencia estructural, aunque por sí sola puede producir cambios de brillo o color. Todo lo anterior fue comprobado mediante experimentación y esta evaluación independiente de funciones de pérdida terminó dándole más peso a la propuesta de Mix Loss como una combinación diseñada para conservar fidelidad píxel a píxel sin descuidar la estructura de la reconstrucción.

Burlin et al. (2017) explora y evalúa de forma comparativa diversas técnicas de inpainting existentes en la literatura utilizando la base de datos CIFAR-10. Entre las arquitecturas implementadas se encuentran los Autocodificadores estándar, los Autocodificadores de eliminación de ruido, modelos basados en Redes Generativas Adversariales (WGAN), modelos basados en densidad como PixelCNN, así como su propia implementación de un modelo de memoria a corto y largo plazo (Long Short-Term Memory, LSTM). Entre sus contribuciones se encuentran algunas técnicas de entrenamiento para mejorar la fidelidad de las reconstrucciones como lo es la implementación de mecanismos de penalización para funciones de pérdida basadas en su-

perposición de parches. En sus experimentos se demuestra que el uso de pérdidas adversariales en el entrenamiento e inferencia mejora significativamente la calidad visual de las reconstrucciones comparadas con la pérdida L2 estándar de inpainting (MSE).

Liu et al. (2018) propone una alternativa robusta y adaptable a las capas convolucionales estándar utilizadas hasta entonces para las tareas de inpainting de imágenes RGB. Su enfoque se conoce como capas convolucionales parciales (partial convolutions) y es particularmente útil para escenarios de inpainting en donde las secciones faltantes de la imagen se presentan en formas irregulares y arbitrarias. De forma general, las capas convolucionales parciales pueden ser consideradas un método de inpainting no ciego, lo cual quiere decir que el modelo tiene la capacidad de distinguir entre aquellos píxeles que se consideran enmascarados o “faltantes” y aquellos que están completos y por lo tanto son “válidos”. A diferencia de las convoluciones tradicionales, donde se utilizan todos los píxeles de la imagen de forma indiscriminada y por lo tanto la salida de cada capa se ve afectada por los píxeles faltantes (inpainting ciego), su propuesta condiciona la operación de convolución únicamente en los píxeles válidos, y esto se ve acompañado de un mecanismo automático de actualización de máscaras el cual da seguimiento al progreso de inpainting capa por capa, esto desemboca en que el tamaño de la máscara de píxeles no válidos se vaya reduciendo paulatinamente hasta producir una imagen completa. Adicionalmente, también se genera una base de datos de máscaras irregulares con más de 12,000 muestras categorizadas por tamaño y por proporción (ratio) de área entre imagen y máscara para permitir la replicación de resultados. Su propuesta, combinada con una arquitectura tipo U-Net fue capaz de superar tanto cuantitativamente como cualitativamente a otros modelos tipo GAN del estado del arte y sus resultados siguen siendo competitivos incluso para estándares actuales.

Asimismo, se señala que en arquitecturas convolucionales estándar el valor utilizado para inicializar las regiones faltantes puede influir considerable-

mente en los artefactos producidos por el modelo. En su comparación, el uso del valor de la media RGB de ImageNet produjo mejores resultados que utilizar un valor constante de 127.5.

Jo y Park (2019) presenta SC-FEGAN, un sistema de edición de rostros humanos basado en GAN en la cual el usuario tiene la posibilidad de dibujar un boceto (sketch) encima del rostro que servirá a modo de máscara de edición, dicha imagen enmascarada será utilizada para entrenar la parte generadora de la GAN, una vez se produzca la salida del generador, la región fuera de la máscara de la imagen original se sobrepondrá y combinará con dicha salida para producir una imagen completa, después de esto se pasará al discriminador de la GAN para determinar si la salida creada por el generador puede o no pasar como una imagen auténtica. El generador está basado en la arquitectura U-Net y el discriminador en SN-patchGAN. SC-FEGAN presenta resultados cualitativos sorprendentes en la edición libre de rostros, desafortunadamente no se presentan sus resultados cuantitativos en el artículo.

Ivanov et al. (2019) introduce una extensión al trabajo de Sohn et al. (2015), los autocodificadores variacionales con condicionamiento arbitrario (Variational Autoencoder with Arbitrary Conditioning, VAEAC). Se trata de un CVAE capaz de condicionar su salida (un subconjunto de características no observadas) "x_b" a partir de cualquier subconjunto arbitrario de características observadas "x_{1-b}" y una máscara binaria "b" la cual indica qué subconjunto de características son observables a la entrada y qué subconjunto se encuentra enmascarado o faltante. Este mecanismo es especialmente útil para las aplicaciones de inpainting e imputación de valores faltantes (missing feature imputation). Su modelo condiciona su distribución a priori basándose en las características observadas y al igual que CVAE utiliza el marco de trabajo Stochastic Gradient Variational Bayes (SGVB) para su entrenamiento. Se demuestra que este tipo de modelos tiene aplicaciones multimodales, ya que es capaz de manejar tanto la generación de valores reales como valores categóricos, produciendo múltiples salidas posibles de

forma no determinística pero realista. Sus experimentos para imputación de datos se llevaron a cabo en la base de datos UCI y aquellos enfocados al inpainting se realizaron en CelebA. Sus resultados fueron competitivos para el estado del arte de ese año e incluso son convincentes para estándares actuales.

Adicionalmente, dentro de la descripción de sus experimentos de inpainting, se retoman distintas estrategias de enmascaramiento de imágenes utilizadas por trabajos previos, incluyendo la máscara tipo Pattern del trabajo de Pathak et al. (2016). En su revisión se explica el proceso de generación de dicha máscara de la siguiente manera: primero se genera una imagen aleatoria de un solo canal con tamaño 600×600, cuyos valores siguen una distribución uniforme; posteriormente, esta imagen se interpola bicúbicamente hasta alcanzar un tamaño de 10000×10000 y se binariza mediante una función escalón de Heaviside con umbral (threshold) de 0.25, considerando como no observados los valores menores a dicho umbral. Para obtener una máscara individual, se recorta aleatoriamente una región de 64×64 dentro de esta imagen binaria y se acepta únicamente si la proporción de píxeles no observados se encuentra entre 20% y 30%; en caso contrario, se toma otro recorte aleatorio. De esta forma se producen regiones irregulares, pero manteniendo cierto control sobre la proporción del área enmascarada.

Lee et al. (2020) introduce MaskGAN, un marco de trabajo de manipulación facial compatible con la edición interactiva de rasgos faciales, esto se logra haciendo uso de máscaras semánticas (la segmentación de un rostro en sus características principales) como una representación intermedia dentro del proceso de edición de imagen. Su sistema está compuesto de 3 componentes principales: La red de mapeo denso (Dense Mapping Network, DMN) la cual se encarga de mapear la máscara semántica manipulada por el usuario a una imagen modificada en el espacio de píxeles producida por una red Generativa Adversarial (GAN); el MaskVAE, un autocodificador variacional que se encarga de modelar el espacio latente de las máscaras semánticas para mantener consistencia dentro de las imá-

genes producidas por el Generador de la GAN; y un mezclador alfa (Alpha Blender), cuyo trabajo consiste en mezclar la imagen original con la imagen modificada para mantener coherencia en las zonas no manipuladas del rostro. Además, introduce la base de datos CelebAMask-HQ, un conjunto de datos basado en CelebA-HQ con más de 30,000 imágenes faciales en alta resolución acompañadas de sus respectivas máscaras semánticas segmentadas.

Zhou et al. (2020) describe el uso de una GAN entrenada con la variación U-Net GAN para realizar el inpainting de rostros humanos con máscaras en posiciones aleatorias, utilizando las bases de datos CelebA-HQ y Flickr-Faces-HQ. Obtiene resultados de 1.46% para su error L1, 26.36 para su Proporción Máxima de Señal a Ruido (PSNR), 0.9107 para su Índice de similitud estructural (SSIM) y 0.0706 para su Similitud Aprendida Perceptual de Parches de Imagen (Learned Perceptual Image Patch Similarity, LPIPS), resultados por encima de las métricas del estado del arte de aquel año, el enfoque de su investigación se basó en igualar las texturas generadas por el inpainting para que fuesen consistentes con las texturas de los rasgos faciales del rostro a completar.

Salem et al. (2021) propone un método GAN en cascada para rellenar regiones aleatorias e irregulares faltantes en un rostro humano. Esto se logra comenzando por una GAN predictora de formas que identifica la estructura del rostro y los puntos clave de sus rasgos faciales. La salida de este predictor de formas se utiliza como entrada “verdadera” para la GAN de inpainting junto con la misma imagen incompleta que se utilizó para entrenar el modelo generativo de la GAN predictora de formas. Sus resultados cualitativos son bastante buenos, desafortunadamente no se presentan resultados cuantitativos en el artículo. La base de datos utilizada fue CelebA-img-align.

Suvorov et al. (2021) introduce LaMa, un marco de trabajo de inpainting diseñado específicamente para el rellenado de imágenes en presencia de máscaras grandes e irregulares, el tipo de corrupción donde a menudo otros

modelos tienen dificultades. Su enfoque y su innovación principal consisten en el uso de convoluciones rápidas de Fourier (Fast Fourier Convolutions, FFCs), un nuevo diseño de capas convolucionales que aprovecha el espacio frecuencial de la transformada de Fourier para aumentar significativamente el tamaño del campo receptivo (receptive field) de la operación de convolución. Las FFCs le permiten al modelo tener información global acerca de la estructura de la imagen desde las primeras capas ocultas, lo cual a su vez disminuye enormemente la necesidad de utilizar redes profundas para obtener la información semántica relevante para la reconstrucción, produciendo mejor generalización con menos parámetros y permitiendo preservar coherencia estructural de forma más natural, incluso en resoluciones altas. Este tipo de modelo compite e incluso es capaz de superar a modelos más pesados en el inpainting de numerosas bases de datos, incluyendo CelebA-HQ, evaluando de forma cuantitativa la distancia de Fréchet de Inception (Fréchet Inception Distance, FID) y la Similitud Aprendida Perceptual de Parches de Imagen (LPIPS).

Kuo et al. (2022) propone un método basado en Deep Learning para la detección automática de zonas dañadas en imágenes antiguas. Su modelo se basa en una arquitectura U-Net reforzada con capas convolucionales tipo bloques densos residuales (Residual Dense Blocks, RDBs), un mecanismo de conexiones intra-capa que combina las fortalezas de ambos bloques residuales con bloques densos mediante concatenación, esto mejora la extracción de características sin incrementar significativamente la cantidad de parámetros. Su modelo toma una imagen dañada a la entrada y produce como salida una máscara binaria que puede ser combinada con cualquier algoritmo de inpainting (de acuerdo con lo reportado, se combina particularmente bien con modelos de inpainting basados en convoluciones parciales), lo cual favorece una metodología modular y controlada para la restauración de imágenes. Su proceso de entrenamiento se llevó a cabo en una base de datos personalizada con anotaciones manuales y técnicas de aumento de datos y los resultados superaron tanto en precisión como en re-

cuperación (recall) a otros modelos convolucionales de extremo a extremo (end-to-end) y a detectores de grietas en pavimento para la misma tarea.

Agarwal y Bhatnagar (2023) presenta nuevos conjuntos de datos modificados a partir de CelebA y CelebA-HQ y evalúa el desempeño de 3 modelos de inpainting del estado del arte aplicados al problema de reconstruir rostros cubiertos por un cubrebocas. Los modelos a evaluar fueron PatchMatch (Barnes C et al., 2010), EdgeConnect (Nazeri K et al., 2019) y Free-Form Image Inpainting with Gated Convolution (Yu J et al., 2019). Sus resultados cuantitativos demostraron que de los 3 modelos Gated Convolution proporciona los mejores resultados para reconstruir el espacio del rostro cubierto por un cubrebocas, teniendo este un Error Absoluto Medio (MAE) de 0.0168, una distancia de Fréchet de Inception (FID) de 2.012, un SSIM de 0.9390 y un PSNR de 28.1843, resultados incluso mejores a aquellos presentados por Zhou et al.

Como puede apreciarse por los trabajos anteriores, existen modelos capaces de rellenar regiones faltantes de una imagen, trabajar con máscaras irregulares y editar rostros a partir de información proporcionada por el usuario (Liu et al., 2018; Jo y Park, 2019; Suvorov et al., 2021). El interés del presente proyecto yace en aprovechar estos avances para desarrollar y ajustar un modelo especializado, buscando un balance entre la calidad de las reconstrucciones y su costo computacional para una tarea concreta. En el caso de los retratos hablados, los rasgos faciales principales ya están definidos y lo que se busca es rellenar los espacios vacíos entre ellos y a su alrededor. Para esta aplicación es necesario determinar qué complejidad de modelo permite obtener reconstrucciones adecuadas sin incrementar demasiado los recursos necesarios para su entrenamiento y ejecución. Por lo anterior, en el presente trabajo se desarrolla y evalúa un modelo convolucional basado en autocodificadores de eliminación de ruido y conexiones tipo U-Net, haciendo uso de máscaras dinámicas y adición de ruido para generar imágenes incompletas durante el entrenamiento. La comparación de distintas configuraciones permite estudiar cómo los

cambios en la arquitectura y los hiperparámetros afectan tanto la calidad de las reconstrucciones como el costo computacional. Esto también permite considerar su uso futuro en equipos con recursos limitados, como las computadoras de placa única (Single-Board Computers, SBCs), cuya capacidad para ejecutar el modelo deberá evaluarse posteriormente.

1.3 JUSTIFICACIÓN

La reconstrucción automática de datos faltantes en rostros humanos mediante inpainting tiene aplicaciones importantes en campos como la seguridad y la justicia, donde frecuentemente se utilizan retratos hablados para identificar a personas. Este proyecto buscará complementar a un sistema de Retratos Hablados mediante sistemas lineales que actualmente está siendo desarrollado en el Centro de Investigación en Ingeniería y Ciencias Aplicadas.

Los resultados del proyecto generarán la metodología y el código necesarios para producir un conjunto de datos orientado al rellenado de las regiones faltantes en retratos hablados y un modelo de referencia los cuales podrán ser utilizados a futuro por otros investigadores para abordar problemas similares o para aplicar estas técnicas en otros contextos como podrían serlo más adelante la combinación de rostros, la modificación de gestos faciales o inclusive la generación de bases de datos faciales. El flujo de trabajo desarrollado en el presente trabajo de investigación puede ser adaptado y reutilizado para el entrenamiento de modelos más complejos o con arquitecturas diferentes a la presentada en este proyecto.

Asimismo, es importante resaltar que a diferencia de otros proyectos de inpainting general o inclusive inpainting facial, la presente investigación cuenta con un enfoque especializado y complementario, por esto mismo el modelo a desarrollar busca optimizarse en cuanto a su tamaño y número de parámetros, resolviendo un problema específico con un coste computacional menor al de los modelos generales.

1.4 HIPÓTESIS

HIPÓTESIS GENERAL

Es posible desarrollar un modelo de inpainting facial basado en redes neuronales convolucionales que sea capaz de reconstruir secciones faltantes de rostros humanos de forma visualmente coherente, utilizando como referencia los rasgos faciales principales presentes en la imagen de entrada.

HIPÓTESIS ESPECÍFICAS

1. La generación de un conjunto de datos facial modificado mediante máscaras dinámicas permitirá simular escenarios de información faltante y entrenar un modelo especializado en reconstrucción facial.
2. El uso de una arquitectura basada en redes neuronales convolucionales, particularmente arquitecturas tipo autocodificador de eliminación de ruido, permitirá extraer características espaciales y semánticas útiles para reconstruir regiones faltantes en imágenes de rostros humanos, manteniendo un costo computacional razonable.
3. La comparación entre distintas configuraciones de arquitectura, función de pérdida e hiperparámetros permitirá identificar una combinación adecuada para mejorar la calidad de las reconstrucciones.
4. La evaluación del modelo mediante métricas de calidad de imagen con y sin referencia combinadas con observación cualitativa permitirá determinar si las reconstrucciones generadas son coherentes visualmente con el rostro original.

1.5 OBJETIVOS

Desarrollar un modelo de inpainting basado en Inteligencia Artificial capaz de rellenar satisfactoriamente las secciones faltantes de un rostro humano a partir de los rasgos faciales principales de una persona, como lo son: los ojos, las cejas, la nariz, la boca, el mentón, el cabello y las orejas de una persona.

OBJETIVOS ESPECÍFICOS

1. Crear al menos un conjunto de datos específico para el entrenamiento del modelo mediante la técnica de aumento de datos, esto se generará a partir de una base de datos existente de rostros humanos como Flickr-Faces-HQ o CelebA-HQ.

2. Desarrollar un flujo de trabajo robusto y expandible en el lenguaje de programación Python para realizar el entrenamiento de los modelos de manera semiautomática y unificada, reduciendo el coste operativo que implicaría realizar dichos entrenamientos en entornos aislados.

3. Desarrollar un programa en Python que describa un modelo de inpainting facial basado en redes neuronales convolucionales con arquitectura de autocodificador de eliminación de ruido y conexiones tipo U-Net.

4. Evaluar el desempeño de las distintas configuraciones de arquitectura, función de pérdida e hiperparámetros de los modelos desarrollados, considerando la calidad de las reconstrucciones y su costo computacional.

5. Realizar un registro detallado de cada uno de los experimentos realizados (experiment logs), así como sus resultados preliminares de forma automática mediante el uso de bases de datos NoSQL y depósitos de almacenamiento (buckets).

6. Analizar el modelo obtenido y aplicar métricas de calidad de imagen para evaluar los resultados tanto de manera cuantitativa como cualitativa.

CAPÍTULO 2: MARCO TEÓRICO

A continuación se presentan los fundamentos teóricos relevantes en los cuales se basa la propuesta metodológica de este trabajo.

2.1 Teoría Fundamental

2.1.1 La Imagen Digital

Una imagen o gráfico digital se define como una representación bidimensional de una imagen mediante una matriz numérica, comúnmente en forma binaria (1s y 0s). Esta representación puede ser estática o dinámica, pudiendo clasificarse como una imagen matricial (o mapa de bits) o como un gráfico vectorial, respectivamente. El mapa de bits es el formato predominante en informática debido a su amplia utilización.

En el contexto de los mapas de bits, y desde una perspectiva matemática, una imagen digital puede ser conceptualizada como una función de dos variables, f(x,y), donde x e y definen una pareja de coordenadas espaciales en un plano, y f en cualquier punto definido por dicho par de coordenadas representa la intensidad o nivel de gris correspondiente en la imagen. En todo mapa de bits existen por lo tanto un número finito de puntos f a los cuales se les conoce con el nombre de píxeles. [(Castillo, 2015), (Gómez,

2017)]

2.1.2 Visión Artificial

La visión por computadora o visión artificial es un campo de la inteligencia artificial (IA) que hace uso de técnicas como el aprendizaje automático (Machine Learning, ML) y las redes neuronales para enseñar a los sistemas computacionales a extraer información relevante de imágenes digitales, videos y otros datos visuales, y tomar recomendaciones o acciones cuando se detectan defectos o problemas. Si la IA capacita a las computadoras para “pensar”, la visión por computadora les permite “ver”, “observar” y “comprender”. [(Castillo, 2015), (IBM, s.f.), (Alvi, 2023)]

2.1.3 Procesamiento Digital de Imágenes (PDI)

El procesamiento digital de imágenes (PDI) engloba un conjunto de técnicas y procedimientos empleados para modificar, procesar o resaltar diversos atributos, valores y características de una imagen digital. También puede implicar el reconocimiento limitado de patrones e, incluso, la restauración o reconstrucción de información codificada o perdida en una imagen, (lo cual es especialmente relevante para nuestros propósitos).

Esta disciplina dentro de las ciencias computacionales guarda una estrecha relación con el procesamiento digital de señales, ya que, en cierto sentido, puede considerarse como una aplicación del procesamiento digital de señales en dos o tres dimensiones.

Por otra parte, en el contexto de la restauración, amplificación y reconstrucción de imágenes, el PDI puede ser considerado como un subcampo de la visión artificial y es, junto con técnicas de Inteligencia Artificial (IA), uno de los métodos utilizados para solucionar el problema del rellenado de imágenes o inpainting.

En otras palabras, el procesamiento digital de imágenes prepara el terreno al mejorar y transformar imágenes digitales, las cuales luego serán interpretadas y comprendidas a través de la visión por computadora. [(Castillo, 2015), (Escalante, 2006), (Domínguez Torres, 1996), (Alvi, 2023)]

2.2 Inpainting

El rellenado de imágenes o inpainting hace referencia originalmente al arte de reparar o reconstruir pinturas o fotografías viejas o dañadas, las cuales tengan a su vez pequeños defectos como lo pueden ser rasgaduras, grietas, polvo, manchas, decoloración o cualquier otro desperfecto que las altere de su apariencia original.

El objetivo del inpainting es entonces corregir dichos defectos a la vez que se intenta mantener la calidad del rellenado para que este sea lo más similar posible al retrato original.

Esta forma de arte ha existido durante varios siglos, siendo realizada esta actividad de forma manual por artistas profesionales y aficionados por igual. Sin embargo, no sería hasta el siglo XVIII cuando se aplicaría por primera vez el método científico a este proceso, siendo Pietro Edwards, el entonces director de Restauración de Imágenes Públicas del Colegio Liberal de Pintura de Véneto en Venecia, Italia el pionero a quien se le atribuye el realizar el proceso de inpainting de forma metódica, estudiando las técnicas del artista original, para así enfocar sus esfuerzos de restauración en transmitir las intenciones originales del artista.

Un par de siglos más tarde, con el surgimiento y evolución de las computadoras y el desarrollo de cada vez más herramientas digitales de edición y técnicas de procesamiento de imagen, se comenzó a presentar un auge en la popularidad del inpainting, al este volverse cada vez más accesible a las masas y digitalizarse.

Una vez generado este interés, la siguiente evolución del inpainting se daría con el resurgimiento de la inteligencia artificial en la forma de nuevas técnicas de aprendizaje automático y deep learning, siendo su desarrollo posible gracias a la evolución de los equipos de cómputo (hardware), la disponibilidad de mayor número de bases de datos cada vez más grandes y los nuevos avances teóricos en algoritmos de entrenamiento.

Es de esta forma que el inpainting se ha convertido en una de las más importantes aplicaciones de la visión artificial, así como un tema tanto interesante como desafiante en el ámbito de la investigación del procesamiento digital de imágenes. [(Elharrouss et al., 2019), (Jam et al., 2021), (Darrow, 2000), (Castillo, 2015)]

2.2.1 Técnicas Tradicionales de Inpainting.

Existen dos principales categorías en las cuales las técnicas de inpainting tradicionales pueden ser clasificadas: los métodos basados en parches (patch-based methods) y los basados en difusión (diffusion-based methods). [(Roy et al., 2021),(Jam et al., 2021)]

2.2.2 Métodos basados en parches.

Comenzando por los métodos basados en parches, estos consisten en rellenar las regiones faltantes de una imagen parche por parche. Esto se realiza mediante la búsqueda iterativa de secciones de la imagen que no se encuentren dañadas (llamadas parches candidatos) y que sean lo suficientemente similares a la región en la que se desee realizar el inpainting.

Estos parches candidatos son entonces seleccionados y copiados en la región correspondiente a rellenar para de esta forma realizar la reconstrucción de la imagen.

Estos algoritmos tienen un buen desempeño a la hora de rellenar imágenes texturizadas, sin embargo, presentan severos inconvenientes para reconstruir imágenes naturales, sobre todo aquellas en las que existan numerosas texturas locales, esto se debe a que tienden a asumir que la textura asociada a una sección no dañada puede ser aplicable al resto de la imagen.

Además, su costo computacional suele ser bastante alto, ya que necesitan calcular un puntaje de similitud para cada posible pareja origen-objetivo de parches. [(Roy et al., 2021)]

2.2.3 Métodos basados en difusión.

Por su parte, los métodos basados en difusión rellenan las secciones faltantes de una imagen mediante la propagación suavizada de los píxeles que se encuentren en el borde de la región faltante (llamada hueco en este contexto).

Los métodos basados en difusión se valen de ecuaciones diferenciales parciales (EDP) a través de las cuales se modela el proceso difusivo para propagar así los colores de los píxeles colindantes a las regiones faltantes o dañadas de una imagen. Dicho acercamiento funciona bien para realizar la reconstrucción de agujeros pequeños en una imagen, pero deja mucho que desear a la hora de reconstruir componentes estructurales o texturas en regiones faltantes más grandes. [(Roy et al., 2021)]

2.3 Inteligencia Artificial

La Inteligencia Artificial (IA) se refiere a una subdisciplina de las ciencias de la computación cuyo objetivo principal consiste en hacer que sistemas artificiales como lo son las computadoras y otras máquinas creadas por el ser humano sean capaces de imitar la inteligencia humana a la hora de realizar tareas diversas y que a su vez sean capaces de mejorar iterativamente mediante el análisis de datos, reconociendo patrones en estos y de esta forma “aprendiendo”.

La aplicación de la IA a una problemática determinada pretende lograr que una tarea que a menudo requiere de la intervención directa de un ser humano para ser realizada sea capaz de automatizarse, buscando obtener resultados lo más cercanos posibles al desempeño que tendría un ser humano o incluso mejores.

En palabras de la Dra. Elaine Rich de la Universidad de Austin Texas (1983):

“La IA es el campo que estudia cómo hacer que las computadoras realicen tareas en las que, de momento, las personas son mejores.”

El término de IA a menudo se utiliza de forma intercambiable con las subdisciplinas que la componen, entre las que se encuentran: El Aprendizaje Automático y El Deep Learning. Sin embargo, es importante resaltar la distinción entre ellos. [(Berzal, 2018), (Rich, 1983)]

2.3.1 Aprendizaje Automático.

El aprendizaje automático (ML) hace referencia al primer gran subconjunto de la IA, el cual le proporciona mecanismos a un sistema computacional para resolver un problema sin la necesidad de programar dicho sistema de forma explícita (Arthur Samuel, IBM). De cierta forma, un sistema computacional basado en aprendizaje automático es capaz de programarse a sí mismo.

Al sistema solo se le provee con un conjunto de datos de entrenamiento y las heurísticas de aprendizaje que el programador haya decidido incluir en su algoritmo de aprendizaje. De esta forma, se busca resolver problemas de tipo NP-difíciles, es decir aquellos problemas para los cuales no existe una forma eficiente de encontrar una solución en un tiempo razonable utilizando una computadora y técnicas de programación tradicionales.

Al concluir el proceso de entrenamiento de un modelo de ML este se comportará de forma similar a una caja negra; al proporcionarle a un modelo entrenado una entrada similar a aquellas con las que se realizó su entrenamiento este será capaz de responder con una salida adecuada, la cual puede tomar muchas formas como lo pueden ser: predicciones, clasificaciones, clustering, sistemas de clasificación, de recomendación, de recuperación de información, etc.

El resultado proporcionado por el algoritmo nunca será perfecto, será una

aproximación aceptable a la solución óptima de dicho problema, sin embargo, esta aproximación será a menudo mejor que aquella que pudiese ser obtenida por técnicas de programación tradicional y el tiempo de ejecución de su algoritmo será a su vez magnitudes de veces menor. [(Berzal, 2018)]

2.3.2 Redes Neuronales.

Las redes neuronales artificiales (RNA), hacen referencia a un modelo matemático y computacional, inspirado en el funcionamiento del sistema nervioso central, más específicamente se puede pensar en ellas como un modelo de IA basado en la estructura del cerebro.

La unidad básica de una RNA es una neurona o nodo, la estructura de cada una de estas unidades de procesamiento está caracterizada por un conjunto de entradas (inputs), las cuales representan conexiones sinápticas con un peso asociado (weight), dicho peso puede ser positivo (excitatorio) o negativo (inhibitorio).

A la suma de estas entradas, siendo multiplicada cada una de ellas por su respectivo peso, se le suma además un valor de sesgo (bias) b (representando el nivel interno de una neurona en reposo), a este producto escalar del peso y las entradas se le conoce como señal de red (net signal).

A esta señal de red se le aplica una llamada función de transferencia (transfer function) o función de activación (activation function) cuyo objetivo comúnmente es: tanto normalizar la escala de los datos como proveer de la característica de no-linealidad a aquello que sería de otra forma una función meramente lineal. Esto último es muy importante ya que, sin esta función de transferencia, nuestra salida (output) solo sería capaz de resolver problemas linealmente separables.

Ahora bien, una neurona por sí misma no es una RNA, si se implementa de

manera independiente se le conoce con el nombre de perceptrón, un modelo de clasificación sencillo que originalmente estaba pensado únicamente para funcionar como un clasificador lineal binario.

Es cuando combinamos varias neuronas, interconectándolas unas con otras a modo de que cada una de ellas funcione como una unidad de procesamiento dentro de una estructura mucho más grande, una RNA, que se pueden realmente obtener resultados para problemas NP-difíciles.

La morfología general para una RNA consiste en dividirla en al menos 3 capas:

1. La “capa de entrada” será aquella que interactúe primero con los parámetros o datos de entrada relevantes al problema que se esté tratando, siendo esta capa el punto de partida de la RNA, generalmente se utilizarán pesos iniciales y funciones de transferencia diferentes a aquellos que serán empleados en otras capas. La cantidad de neuronas en esta capa depende de la cantidad de características o parámetros que tenemos disponibles respecto al conjunto de datos de entrada a partir del cual se pretende obtener un resultado.

2. La segunda capa, viniendo inmediatamente después de la capa de entrada y aquella cuyas entradas serán las salidas de esta, es conocida como la “capa oculta”. Será aquí donde se realizarán la mayor cantidad de cálculos en el caso de las RNAs más complejas, es importante recalcar que es en esta capa donde generalmente se encuentra la mayor concentración de neuronas y que puede haber múltiples capas ocultas en una sola RNA. Además de ser el “centro computacional” de las llamadas redes neuronales profundas (DNN) , son estas capas intermedias las que permiten a una RNA abstraer características, parámetros y representaciones de datos complejos a partir de los datos de entrada iniciales, dando así robustez y capacidad de generalización al sistema. La cantidad de neuronas en esta capa es muy

variable, pero de este hiperparámetro dependerá en gran medida el desempeño de la RNA.

3. Finalmente, la “capa de salida”, como su nombre lo indica, se refiere a la última capa de una RNA, aquella cuyas entradas serán las salidas de la última capa oculta y la que se asegurará de normalizar los valores de las salidas que se pretenden obtener como resultado final. La cantidad de neuronas en esta capa dependerá de la cantidad de características o parámetros que se necesitan adquirir a la salida, y variará de acuerdo con la naturaleza del problema.

Es mediante la utilización de técnicas como el descenso de gradiente (gradient descent) y la retropropagación (backpropagation), que los modelos de RNA se ajustan para mejorar la precisión de sus predicciones con cada época de entrenamiento. De esta forma se puede decir que las RNA “aprenden”. [(Berzal, 2018), (Fullér, 2000), (DeepAI, s.f.), (IBM, 2021)]

2.3.3 Deep Learning (DL)

El deep learning se refiere a un subconjunto del aprendizaje automático que abarca a toda RNA con tres o más capas. Dicha topología de RNA busca imitar la capacidad del cerebro humano para aprender de grandes volúmenes de datos, aunque en la actualidad todavía no alcanza la capacidad de su contraparte orgánica. Mientras que las RNAs de una sola capa (los perceptrones) solo pueden hacer predicciones aproximadas muy limitadas, las RNA con capas adicionales (perceptrones multicapa) mejoran considerablemente la precisión de sus resultados y el refinamiento de sus predicciones.

Este enfoque impulsa numerosas aplicaciones de inteligencia artificial, automatizando tareas analíticas y físicas sin intervención humana. Lo cual lo ha vuelto fundamental en varios productos y servicios cotidianos, como

asistentes digitales y detección de fraudes, así como en tecnologías emergentes como lo son los vehículos autónomos.

El deep learning simplifica el proceso previo al entrenamiento en cuanto a la preparación y normalización de los datos de entrada, permitiendo que los algoritmos procesen datos no estructurados como texto e imágenes.

Los modelos de DL automatizan la extracción de características e incluso son capaces de generar características nuevas, cada vez más abstractas, a partir de las características existentes, reduciendo enormemente la necesidad de la intervención humana.

Por ejemplo, al clasificar fotos de animales, un modelo de DL es capaz de identificar características clave, como las orejas, para así ser capaz de distinguir entre diferentes animales. Este proceso elimina la necesidad de que un experto humano establezca manualmente la jerarquía de características, como suele ser necesario en el aprendizaje automático. [(Berzal, 2018), (DeepAI, s.f.), (IBM, s.f.)]

2.3.4 Redes Neuronales Convolucionales (CNN)

Las Redes Neuronales Convolucionales (CNN por sus siglas en inglés) constituyen una subcategoría de RNA y son actualmente uno de los modelos más reconocidos para la clasificación de imágenes debido a su excepcional rendimiento en este campo. Las CNNs aprenden más rápido y logran tasas de error mucho más bajas que otros modelos de deep learning en este dominio.

Las CNNs inspiran su morfología en la naturaleza, ya que la conectividad entre las neuronas artificiales de la misma se asemejará a la organización de la corteza visual animal.

Su método de operación puede parecer sencillo a grandes rasgos: el usuario proporciona una imagen de entrada matricial (mapa de bits). Dicha

imagen consistirá normalmente de tres dimensiones:

- Dos dimensiones para la imagen en escala de grises.
- Una tercera dimensión con una profundidad de 3 para representar los colores fundamentales RGB (rojo, verde, azul).

A diferencia de un modelo estándar de Perceptrón Multicapa (Multilayer Perceptron, MLP), que consiste únicamente en un módulo de clasificación, la arquitectura estándar de las CNN se basa en dos módulos principales:

1. Módulo Convolucional: Su objetivo es extraer características específicas de cada imagen de entrada al comprimirlas para reducir su tamaño original. En resumen, las imágenes de entrada son sometidas a una serie de filtros, creando nuevas imágenes llamadas mapas de características (feature maps). Estos mapas de características obtenidos son concatenados en un vector de características llamado el código CNN. El código CNN no solo puede ser utilizado en la clasificación de imágenes (a pesar de que esta es la tarea más común para la que es usado), sino también en otras potenciales aplicaciones como la transferencia de estilo neuronal o por supuesto el inpainting.
2. Módulo de Clasificación: El código CNN obtenido como salida del módulo convolucional es utilizado para alimentar al segundo módulo, el cual consta de varias capas de neuronas artificiales interconectadas, cuyo conjunto es conocido como un Perceptrón Multicapa (MLP). El papel de este módulo es combinar las características del código CNN para así llevar a cabo la clasificación correcta de las imágenes. [(Berzal, 2018), (DataScientest, 2023), (Saha, 2018), (IBM, s.f.)]

2.3.5 Funciones de transferencia

Las funciones de transferencia, también conocidas como funciones de activación, son, en el contexto de las redes neuronales artificiales, funciones matemáticas que calculan la salida de una neurona o capa a partir de su señal de red. En redes neuronales profundas, Maas et al. describen funciones como $tanh$, ReLU y Leaky ReLU como transformaciones no lineales aplicadas punto a punto sobre las unidades ocultas de una red. Su importancia yace en que introducen no linealidad dentro del modelo, permitiéndole representar relaciones más complejas que una transformación meramente lineal. Visto de otra forma, se podría decir que sin la aplicación de funciones de transferencia a una red neuronal, estos modelos solamente serían capaces de resolver problemas lineales o linealmente separables (Complejidad P). [(MathWorks, s.f.), (Cybenko, 1989), (Maas et al., 2013)]

2.3.5.1 Tangente Hiperbólica

Una de las funciones de transferencia más utilizadas en redes neuronales es la tangente hiperbólica, comúnmente escrita como $tanh$ o $tansig$, la cual pertenece a la categoría de las funciones de transferencia sigmoidales (aquellas cuya curva característica tiene forma de S). Esta función transforma valores reales al intervalo $[-1, 1]$ de forma simétrica, por lo que resulta útil cuando se desea producir activaciones acotadas y centradas alrededor de cero. MathWorks define $tansig$ como una función matemáticamente equivalente a $tanh$, expresada de la siguiente forma:

$$f(x) = \text{tanh}(x) = \frac{2}{1 + e^{-2x}} - 1$$

Una característica importante de $tanh$ es que sus salidas pueden tomar valores negativos, nulos o positivos, lo cual la vuelve más balanceada que la mayoría de sigmoides. Sin embargo, al acercarse a sus valores asintóticos

-1 y 1, la función puede saturarse. En redes profundas, esta saturación puede contribuir al problema del desvanecimiento del gradiente (vanishing gradient problem), ya que las capas inferiores pueden recibir gradientes cercanos a cero y el entrenamiento puede volverse más lento o inestable a causa de ello. [(MathWorks, s.f.), (Maas et al., 2013)]

2.3.5.2 ReLU

La Unidad Lineal Rectificada, mejor conocida como ReLU por sus siglas en inglés, es una función de activación por tramos que conserva los valores positivos de la señal de red y convierte sus valores negativos en cero:

$$f(x) = \mathsf{max}(0, x)$$

De forma equivalente:

$$f(x) = \begin{cases} x, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

La principal ventaja de ReLU es que, mientras la unidad (capa o neurona) se encuentra activa, su derivada es igual a 1, lo cual ayuda a evitar el desvanecimiento del gradiente en redes profundas y vuelve su cálculo computacionalmente sencillo. Sin embargo, cuando la entrada es negativa, el gradiente se vuelve cero, por lo que algunas unidades pueden dejar de actualizar sus pesos al permanecer inactivas durante el entrenamiento. [(Maas et al., 2013), (Nair & Hinton, 2010)]

2.3.5.3 Leaky ReLU

Leaky ReLU surge como una modificación directa de ReLU. En lugar de asignar cero a todos los valores negativos, conserva una pendiente pequeña y fija para dichos valores de entrada:

$$f(x) = \begin{cases} x, & x > 0 \\ \alpha x, & x \leq 0 \end{cases}$$

donde α suele definirse como una constante pequeña. En la formulación evaluada por Maas et al., por ejemplo, se utiliza un valor de pendiente negativa de 0.01. Esto permite que las unidades reciban un gradiente distinto de cero y que sus pesos se sigan actualizando durante el entrenamiento, ya que los valores negativos dejan de ser anulados completamente. [(Maas et al., 2013), (He et al., 2015)]

2.3.5.4 Parametric ReLU

Parametric ReLU (PReLU) extiende esta misma idea, pero reemplaza la pendiente negativa fija por un parámetro aprendible durante el entrenamiento:

$$f(x) = \begin{cases} x, & x > 0 \\ ax, & x \leq 0 \end{cases}$$

o, de forma equivalente:

$$f(x) = \max(0, x) + a \min(0, x)$$

Si $a = 0$, la función se comporta como ReLU; si a es una constante pequeña,

se comporta como Leaky ReLU; y si a se optimiza junto con el resto de los pesos de la red, se trata de PReLU.

El parámetro aprendible de PReLU puede compartirse con distintos niveles de granularidad. La primera posibilidad consiste en utilizar un único parámetro aprendible global para toda la red. Una segunda posibilidad yace en compartir un parámetro por capa, de modo que todas las unidades (neuronas o canales) de una misma capa utilicen la misma pendiente negativa. Finalmente (en el caso de redes neuronales convolucionales), la formulación por canal propuesta por He et al. permite aprender un parámetro distinto por cada canal de activación o mapa de características, agregando una cantidad reducida de parámetros adicionales al modelo. Por todo lo anterior, dependiendo de cómo se utilice, PReLU permite aumentar la expresividad de la no linealidad sin modificar sustancialmente la arquitectura base de una red neuronal. [(He et al., 2015)]

2.3.6 Redes Generativas Adversariales (GAN)

Las Redes Generativas Adversariales (GAN) se refieren a una clase de modelo de deep learning surgida a partir de los trabajos de Jürgen Schmidhuber en los años 90. A pesar de existir durante ya varias décadas, el término sólo fue popularizado hasta que Ian Goodfellow publicó su artículo ”Generative Adversarial Nets” en el año 2014. Los modelos GAN son altamente reconocidos en el ámbito de clasificación de imágenes debido a que presentan uno de los mejores rendimientos al solventar este tipo de problemáticas.

Para entender las GAN, es fundamental comprender dos términos principales: ”generativo” y ”adversarial”. ”Generativo” se refiere a la capacidad de producir algo, como transformar una imagen de entrada en una salida modificada, por ejemplo, convertir la imagen de un caballo en la de una cebra. Por otro lado, ”adversarial” implica enfrentar a dos elementos, en

el caso de las GAN, se enfrenta el resultado generativo (imágenes falsas) contra un conjunto de imágenes reales mediante un mecanismo llamado discriminador.

Goodfellow ilustró la dinámica de las GAN con una analogía: el modelo generativo se comportaría como un equipo de falsificadores tratando de producir billetes falsos sin ser detectados, mientras que el discriminador tomaría el papel de la policía tratando de detectar las falsificaciones. Esta competencia impulsará a ambas partes a mejorar constantemente hasta que las falsificaciones lleguen a ser indistinguibles de los billetes auténticos.

En términos prácticos, las GAN pueden generar ejemplos falsos que se asemejen a imágenes reales de números, personas, animales, etc. El generador, alimentado con imágenes de entrada aleatorias (o ruido), generará muestras que se esperará se parezcan a parte del conjunto de datos real, mientras que el discriminador evaluará si dichas muestras son reales o falsas.

La optimización de las GAN es un desafío, ya que ambos modelos deben mejorar constantemente a la par. El generador buscará generar falsificaciones más realistas, mientras que el discriminador buscará distinguir entre imágenes reales y falsas con mayor precisión. El equilibrio se alcanzará solamente cuando el discriminador no sea capaz de distinguir entre las imágenes reales y falsas en un 50% de los casos. [(Hansen, 2022), (Jo & Park, 2019)]

2.4 Aumento de Datos

El aumento de datos describe un conjunto de técnicas empleadas en deep learning para suplementar conjuntos de datos “incompletos”, esto se logra mediante la generación y adición de puntos de datos o muestras faltantes los cuales contribuirán a aumentar la capacidad de análisis de un conjunto de datos preexistente.

De forma más específica esto se manifiesta a través de la generación de copias modificadas de datos preexistentes en el conjunto, con esto se busca incrementar el tamaño y diversidad del mismo sin comprometer la consistencia ni la coherencia de los datos.

De acuerdo con el Dr. Jacob Murel (IBM) “los datos aumentados pueden entenderse como el suministro artificial de datos potencialmente ausentes del mundo real.”.

El uso de esta técnica está asociado a mejoras en la optimización y generalización del modelo al que es aplicada, ya que tener una mayor cantidad y variedad de datos de entrenamiento a menudo resulta en un modelo más robusto que no es tan sensible al problema de sobreajuste.

Es importante mencionar que muchos marcos de trabajo de deep learning como Pytorch, Keras y Tensorflow contienen funciones para el aumento de datos, principalmente aplicado a la generación dinámica de conjuntos de datos de imágenes. En lo que se refiere al aumento de datos aplicado a imágenes, sus técnicas específicas a menudo son clasificadas en las siguientes categorías:

- Transformaciones Geométricas: Denotan técnicas que alteran la imagen a nivel espacial e incluyen, pero no se limitan a transformaciones como el cambio en el nivel de zoom, el cambio de tamaño de la imagen, y los cambios de orientación (rotación o espejo de la imagen).

- Transformaciones Fotométricas: Denotan técnicas que alteran los canales RGB de la imagen e incluyen, pero no se limitan a transformaciones como el ajuste de brillo, el ajuste de saturación y la escala de grises de la imagen.
- Transformaciones Excluidas: Denotan aquellas técnicas que no pueden ser clasificadas por ninguna de las dos categorías anteriores e incluyen, pero no se limitan a transformaciones como el filtrado de núcleo (kernel filtering; nitidez o desenfoque de la imagen), el recorte aleatorio (random cropping) de secciones de la imagen para ser usadas como una imagen nueva, la eliminación aleatoria (random erasing) de secciones de la imagen, y la inyección de ruido (píxeles aleatorios negros, blancos o de color en una imagen de acuerdo a una distribución gaussiana).

Finalmente debe tenerse en cuenta que la técnica de aumento de datos es fundamentalmente diferente a otras técnicas similares usadas para expandir conjuntos de datos preexistentes, entre las que resalta particularmente la generación de datos sintéticos. La diferencia entre ambas técnicas radica en que los datos sintéticos describen la generación automática de datos completamente artificiales, mientras que el aumento de datos describe la generación de copias modificadas de datos preexistentes.

De acuerdo con el estado del arte actual, el aumento de datos es una técnica establecida que consistentemente ha probado mejorar el desempeño de diversos modelos de inteligencia artificial, mientras que el uso de datos sintéticos es relativamente nuevo en este ámbito por lo que sus efectos en el entrenamiento siguen siendo estudiados hasta la fecha. [(IBM, 2024)]

2.5 Autocodificadores

Los autocodificadores son un tipo de arquitectura de red neuronal cuyo diseño se enfoca a la compresión y descomposición (codificación) de datos de entrada a sus características más abstractas y esenciales, con el objetivo de más adelante reconstruir dichos datos de entrada de vuelta a su forma original (decodificar) a partir de su representación codificada.

La morfología de los autocodificadores les permite descubrir "variables latentes" en los datos de entrada que se proporcionen durante su entrenamiento; esto se refiere a aquellas variables ocultas o aleatorias las cuales a pesar de no ser directamente observables representan de forma fundamental como se distribuyen los datos. En otras palabras, las variables latentes son las características más comprimidas y abstractas que describen los patrones fundamentales de nuestros datos de entrada.

Al conjunto de todas las variables latentes producidas por un autocodificador se le conoce con el nombre de "espacio latente" y es a partir de este espacio latente que el autocodificador aprenderá a reconstruir los datos originales.

Existen numerosas variantes de autocodificadores las cuales son utilizadas para una gran variedad de aplicaciones, pero en general todos los autocodificadores compartirán algunos elementos estructurales básicos:

- El codificador: Está conformado por aquellas capas de la red neuronal cuya tarea es codificar los datos de entrada que se proporcionen a una representación comprimida de los mismos. Este proceso se realiza mediante una reducción de dimensionalidad obtenida progresivamente al disminuir la cantidad/dimensión de nodos/neuronas disponibles para que fluyan los datos a medida que estos se adentran a capas más profundas de la red. Esto fuerza a que los datos de entrada se compriman en menos dimensiones, descartando información irrelevante.

- El cuello de botella (bottleneck) o “código”: Está conformado por la capa que contiene la representación más comprimida de los datos de entrada, es decir el espacio latente con la menor dimensionalidad posible. Esta capa no solo servirá como la capa de salida del codificador, sino también como la capa de entrada del decodificador. Es importante mencionar que el tamaño de este cuello de botella es un aspecto fundamental del diseño de todo autocodificador, ya que en los modelos más eficientes este será determinado por el número mínimo de variables latentes necesarias para llevar a cabo la reconstrucción de los datos de entrada y a su vez determinará no solo el nivel de abstracción del espacio latente sino también la capacidad de generalización del modelo.

- El decodificador: Está conformado por una serie de capas ocultas con una cantidad/dimensión progresivamente más grande de nodos/neuronas a través de los cuales se irán reconstruyendo los datos de entrada de vuelta a su estado pre-codificación hasta llegar a la dimensionalidad original. Estos datos de salida decodificados serán entonces comparados de manera automática con la “verdad fundamental” (ground truth), es decir con los datos de entrada originales para calcular el error de reconstrucción y a partir de este entrenar el modelo. Esto último convierte a los autocodificadores en modelos de “aprendizaje autosupervisado”, es decir que son capaces de soportar conjuntos de datos no etiquetados lo cual los convierte en modelos altamente robustos y flexibles. [(IBM, 2023)]

2.6 Autocodificadores de eliminación de ruido

Una variante particularmente interesante de los autocodificadores es la de los autocodificadores de eliminación de ruido, como su nombre sugiere, a este tipo de autocodificadores se les proporcionan datos parcialmente corrompidos como entrada y se les entrena para remover información irrelevante mediante reducción de dimensionalidad, esto los vuelve capaces de reconstruir los datos de entrada sin corrupción.

Esta variante resalta con otros autocodificadores en el sentido de que no se le proporcionan los datos con la “verdad fundamental” a la entrada, sino que se le dan los datos originales con ruido (generalmente de tipo Gaussiano) agregado. El ejemplo más común de esto es con imágenes, a las cuales se les agrega estática aleatoria, y el autocodificador de eliminación de ruido aprenderá a ignorar el ruido y centrarse en reconstruir la imagen original a partir del espacio latente. Durante el proceso de entrenamiento, la salida del decodificador no será comparada nuevamente con la imagen corrompida para calcular el error, sino con la imagen original sin corrupción. [(IBM, 2023)]

2.7 Arquitectura U-Net

U-Net pertenece a la familia de arquitecturas codificador-decodificador, por lo que puede considerarse como estructuralmente cercana a los autocodificadores descritos anteriormente. En ambos casos, el codificador transforma los datos de entrada en una representación de menor dimensionalidad y mayor abstracción, mientras que el decodificador reconstruye una salida a partir de dicha representación. Sin embargo, esta semejanza no implica que toda U-Net sea un autocodificador, ya que los autocodificadores suelen estar definidos por el objetivo de reconstruir su propia entrada, mientras que una arquitectura tipo codificador-decodificador puede ser entrenada para producir una salida completamente distinta. En su aplicación original, U-Net fue propuesta para la segmentación semántica de imágenes biomédicas, por lo que la salida esperada no era la imagen de entrada reconstruida, sino una predicción localizada a nivel de píxel. [(IBM, 2023), (Ronneberger et al., 2015)]

La arquitectura U-Net original está compuesta por una ”ruta de contracción” y una ”ruta de expansión” aproximadamente simétrica. La ruta de contracción cumple una función equivalente a la del codificador: reduce progresivamente la resolución espacial de la imagen mediante capas convolucionales y operaciones de reducción, con el objetivo de capturar información semántica y características cada vez más abstractas. La ruta de expansión cumple por su parte con la función del decodificador: recupera progresivamente la resolución espacial original mediante operaciones de aumento de tamaño, como convoluciones transpuestas (transposed convolutions), y genera una salida densa en la que cada región de la imagen conserva una correspondencia espacial con la entrada. [(Ronneberger et al., 2015), (MathWorks, s.f.)]

La principal diferencia estructural entre una U-Net y un codificador-decodificador convencional yace en sus conexiones de salto (skip

connections). En un autocodificador simple, la información debe atravesar todas las capas de la red, incluyendo el cuello de botella, antes de llegar al decodificador. El cuello de botella es útil para forzar la extracción de variables latentes y evitar que el modelo copie la entrada de forma trivial, pero también puede eliminar información espacial o semántica fina que puede resultar relevante en problemas de imagen. Las conexiones de salto de U-Net permiten transferir mapas de características desde una capa del codificador hacia la capa correspondiente del decodificador, generalmente mediante concatenación, de modo que la reconstrucción no dependa únicamente de la representación comprimida. [(IBM, 2023), (Ronneberger et al., 2015)]

Esta propiedad es especialmente relevante en tareas de transformación de imagen a imagen, donde la entrada y la salida suelen compartir información de bajo nivel, como la posición de bordes, contornos y estructuras locales. Isola et al. señalan que, en un codificador-decodificador tradicional, obligar a que toda esta información atraviese el cuello de botella puede ser una limitación; por ello incorporaron conexiones entre la capa i del codificador y la capa $n-i$ del decodificador, siguiendo la forma general de U-Net. En sus experimentos, remover dichas conexiones convierte la U-Net en un codificador-decodificador convencional, mientras que conservarlas mejora la calidad de los resultados generados. [(Isola et al., 2017)]

2.8 Inpainting ciego y no ciego

Los métodos de inpainting pueden clasificarse como ciegos o no ciegos dependiendo de si la región dañada de la imagen se define de forma explícita antes de realizar la reconstrucción. En el inpainting no ciego, la zona faltante a rellenar se le proporciona al modelo mediante información adicional en la entrada, esto se hace comúnmente a través de una máscara binaria que indica qué píxeles deben ser reconstruidos. Por otra parte, en el inpainting ciego, dicha información no se encuentra disponible, por lo que el modelo debe inferir la localización de la región dañada antes o durante el proceso de reconstrucción. Esto vuelve al inpainting ciego un problema más complejo, ya que el sistema debe ser capaz de resolver la localización de la corrupción y la generación del contenido faltante simultáneamente. [(Wang et al., 2020), (Phutke et al., 2023)]

Una estrategia común para tratar el problema del inpainting ciego consiste en introducir una etapa intermedia de estimación de máscara al modelo. En otras palabras, el problema puede formularse como un proceso de dos partes: estimar la región de la imagen a rellenar y posteriormente generar el contenido correspondiente. Sin embargo, el usar esta estructura de dos pasos introduce a su vez una posible fuente de error, ya que si una máscara se estima de manera incorrecta esto puede propagar errores hacia la etapa de inpainting y producir artefactos visuales. Por esta razón, los métodos ciegos suelen requerir mecanismos adicionales para controlar la incertidumbre de la máscara estimada y reducir el impacto de los errores en la localización de la misma. [(Wang et al., 2020), (Phutke et al., 2023)]

En el caso del inpainting no ciego, una máscara binaria puede incorporarse directamente como parte de la entrada de la red. Si la red recibe una imagen corrupta junto con una máscara binaria que indica la región faltante, le será más sencillo procesar huecos con posiciones, tamaños y formas variables. De esta forma, la máscara funciona como una guía espacial que separa las

regiones conocidas de aquellas que deben ser reconstruidas, permitiendo que el modelo utilice el contenido válido de la imagen como referencia contextual. [(Yu et al., 2018)]

Es importante mencionar que la convención utilizada para representar una máscara puede cambiar entre implementaciones. Algunos trabajos le asignan el valor de 1 a los píxeles corruptos y 0 a los píxeles válidos, mientras que otros hacen lo opuesto. [(Yu et al., 2018), (Phutke et al., 2023)]

2.9 Métricas y funciones de pérdida para inpainting

En el caso específico del inpainting mediante redes neuronales, las métricas de calidad de imagen y las funciones de pérdida se encuentran estrechamente relacionadas, pero cumplen papeles distintos dentro de un sistema de reconstrucción. Algunas métricas pueden utilizarse como criterio de entrenamiento, mientras que otras serán delegadas únicamente como herramientas de evaluación posterior, y algunas más pueden cumplir con ambas funciones dependiendo de su implementación. Durante el proceso de entrenamiento, los valores de estas medidas guían la actualización de los pesos internos del modelo; mientras que en la evaluación de resultados, permiten comparar cuantitativamente la imagen reconstruida contra una referencia o contrastar los resultados obtenidos con aquellos reportados en otros trabajos. A continuación se presentan las métricas más comunes para medir la calidad de una reconstrucción de imagen.

2.9.1 Error L1:

También conocido como el Error Absoluto Medio (MAE), esta métrica mide la diferencia media absoluta entre los valores predichos por el modelo y los valores reales de referencia. Su valor siempre es positivo y no distingue entre estimados que sean demasiado bajos o altos. Su ecuación consiste en la sumatoria del valor absoluto de todos los errores (diferencia entre valor predicho y valor real de cada píxel) dividido entre el número de muestras (cantidad de píxeles):

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

El error L1 presenta un comportamiento robusto ante la presencia de valores atípicos (outliers), sin embargo, tiende a sobre penalizar errores pequeños a comparación de otras métricas. Es considerado una métrica con referencia (Bergmann y Stryker, 2024)

2.9.2 Error L2:

También conocido como el Error Cuadrático Medio (MSE), esta métrica mide la diferencia media al cuadrado entre los valores predichos por el modelo y los valores reales de referencia. Su valor siempre es positivo por lo que al igual que el error L1 solo toma en cuenta la magnitud del error y no su dirección. Su ecuación consiste en la sumatoria del valor al cuadrado de todos los errores (diferencia entre valor predicho y valor real de cada píxel) dividido entre el número de muestras (cantidad de píxeles):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

El error L2 presenta un comportamiento desproporcionadamente sensible a valores atípicos debido a que el valor cuadrado de un error grande resultará en un valor enorme. Sin embargo, a diferencia del error L1, penaliza muy poco a errores pequeños (valores menores a 1). Es considerado una métrica con referencia (Bergmann y Stryker, 2024)

2.9.3 Proporción máxima de señal a ruido (PSNR):

En español la Proporción Máxima de Señal a Ruido es una métrica que mide de forma logarítmica la relación entre el valor máximo posible de ruido en una señal y el ruido presente en dicha señal el cual afecta la calidad de su representación. En otras palabras, esta métrica permite calcular en decibeles qué tanto mejor es la proporción de ruido presente en la señal

respecto al peor caso posible. En el contexto de imágenes este concepto es aplicado ampliamente, en este caso el valor máximo posible tiende a ser 255 para un píxel dado (1 si el valor está normalizado), y se mide su relación respecto a la raíz cuadrada del MSE.

Su ecuación está dada por:

$$\text{PSNR} = 20 \log_{10} \left(\frac{MAX_f}{\sqrt{MSE}} \right)$$

Donde:

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (y_{i,j} - \hat{y}_{i,j})^2$$

MAX_f es el valor máximo que puede presentar un píxel en la imagen.

Una forma un poco más clara de interpretar estos valores puede observarse en la siguiente tabla:

PSNR (dB)	**Calidad Visual**	**Interpretación cualitativa**	**Interpretación Cuantitativa**
> 40 dB	Excelente	Casi indistinguible del original	$> 10{,}000\times$ mejor que el peor caso posible
30 dB	Buena	Poca distorsión, generalmente aceptable	$\approx 1{,}000\times$ mejor que el peor caso posible
20 dB	Mala	Distorsión notable, pérdida de detalle	$\approx 100\times$ mejor que el peor caso posible
< 20 dB	Muy mala	Degradación severa	$< 100\times$ mejor que el peor caso posible

Es importante mencionar que al usar el MSE en su cálculo el PSNR se considera una métrica de calidad de imagen con referencia. Y que el peor caso posible sería que un píxel blanco RGB 255 fuera considerado uno negro RGB 0 por el modelo. (National Instruments, 2024)

2.9.4 Structural Similarity Index Measure (SSIM):

En español el índice de similitud estructural es una métrica de calidad de imagen con referencia inspirada en el sistema de percepción visual humano, está diseñado para emular la forma en la que los seres humanos perciben la calidad visual de una imagen y consta de 3 componentes principales:

- Luminosidad
- Contraste
- Estructura

La luminosidad de una imagen se calcula en base a qué tan similares son las intensidades promedio de la imagen predicha respecto a la imagen original. Su ecuación es:

$$L(x,y) = \left(\frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \right)$$

Donde

- μ_x es el valor promedio sobre una ventana (generalmente de 11x11 píxeles) en la imagen original
- μ_y es el valor promedio sobre una ventana (generalmente de 11x11 píxeles) en la imagen predicha
- C_1 es una constante arbitraria para evitar la división sobre 0

El contraste de una imagen se calcula en base a qué tan similares son las desviaciones estándar promedio de la imagen predicha respecto a la imagen original. Su ecuación es:

$$C(x,y) = \left(\frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \right)$$

Donde

- σ_x es la desviación estándar sobre una ventana (generalmente de 11x11 píxeles) en la imagen original
- σ_y es la desviación estándar sobre una ventana (generalmente de 11x11 píxeles) en la imagen predicha
- C_2 es una constante arbitraria para evitar la división sobre 0

La similitud estructural de una imagen se calcula en base a la correlación de la imagen predicha respecto a la imagen original. Su ecuación es:

$$S(x,y) = \left(\frac{\sigma_{xy} + C_3}{\sigma_x \sigma_y + C_3} \right)$$

Donde

- σ_{xy} es la covarianza sobre una ventana (generalmente de 11x11 píxeles) entre la imagen original y la imagen predicha
- C_3 es una constante arbitraria para evitar la división sobre 0

De esta forma, el SSIM puede ser definido como:

$$\text{SSIM}(x,y) = L(x,y)^{\alpha} \cdot C(x,y)^{\beta} \cdot S(x,y)^{\gamma}$$

Generalmente $\alpha = \beta = \gamma = 1$ y $C_3 = \frac{C_2}{2}$ por lo que la ecuación final se simplifica a:

$$\text{SSIM}(x,y) = \left(\frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \right)$$

El rango de valores más típico para la métrica SSIM se encuentra entre 0 y 1 donde 1 indica una similitud estructural perfecta y 0 indica una similitud estructural nula. (National Instruments, s. f.) (MathWorks, s. f.)

2.9.5 Distancia de Fréchet de Inception (FID):

La distancia de Fréchet de Inception es una métrica de calidad de imagen con referencia que ayuda a cuantificar el realismo y la diversidad de las imágenes generadas por una red neuronal convolucional (generalmente se emplea en GANs).

A diferencia de PSNR y SSIM, FID no mide las diferencias entre la imagen real y la imagen predicha en el espacio de píxeles, sino que lo hace en el espacio abstracto de características que se genera en los filtros de una red neuronal convolucional, los mapas de características intra-capa de la red neuronal Inception-v3 de Google. Esto nos da información importante de la similitud semántica entre ambas imágenes, su contenido, su estilo y su variación.

FID trata a ambas imágenes como si fuesen muestras de distribuciones Gaussianas multivariantes dentro del espacio latente de las capas más profundas de Inception-v3, a partir de esto calcula la distancia de Fréchet entre ambas distribuciones.

Los pasos para calcular FID son más complejos que las anteriores métricas mencionadas, sin embargo, pueden resumirse de la siguiente manera:

Paso 1: Extraer las características abstractas usando la red neuronal Inception-v3, esto nos dará 2 conjuntos de matrices de características:

$\{f_1, f_2, f_3, \ldots, f_n\}$ para la imagen real

$\{\hat{f}_1, \hat{f}_2, \hat{f}_3, \ldots, \hat{f}_m\}$ para la imagen predicha/generada

Donde cada valor f_i representa un vector de características abstractas (ejemplo $f_1 = [0.1, 0.3, 0.9]$)

Paso 2: Para modelar los conjuntos de matrices de características como distribuciones Gaussianas, se calculan la media multivariante (un

vector) y la covarianza multivariante (una matriz) para cada conjunto:

μ_r y Σ_r para el conjunto real

μ_g y Σ_g para el conjunto predicho/generado

Paso 3: Calcular la distancia de Fréchet entre las 2 Gaussianas, donde:

$$FID = \|\mu_r - \mu_g\|^2 + TR\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}\right)$$

Donde:

- $\|\mu_r - \mu_g\|^2 = (\mu_r - \mu_g)^T \cdot (\mu_r - \mu_g)$ es la norma vectorial euclídea entre ambas medias vectoriales
- $TR\left(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}}\right)$ es la traza (suma de los elementos diagonales) de la matriz de covarianzas

De esta manera FID mide no solamente la distancia entre los puntos medios de ambas distribuciones Gaussianas, sino que también la diferencia entre las formas de las distribuciones.

Para el caso de esta métrica un valor menor indica una mejor calidad de reconstrucción/generación. Un valor de FID de 0 indicaría que la imagen real y la imagen predicha/generada tienen distribuciones idénticas. Por otra parte, un valor de FID mayor a 100 indicaría que las distribuciones son extremadamente disimilares. (Lawton, s. f.) (Heusel et al., 2018)

2.9.6 Similitud perceptual aprendida de parches de imagen (Learned Perceptual Image Patch Similarity, LPIPS):

En español la Similitud Aprendida Perceptual de Parches de Imagen es una métrica de calidad de imagen con referencia capaz de medir la similitud

perceptual entre dos imágenes o parches de imagen. A diferencia de métricas como L1, L2, PSNR o SSIM, LPIPS no compara directamente los valores de los píxeles ni estadísticas estructurales locales, sino que calcula la distancia entre representaciones profundas extraídas por una red neuronal convolucional preentrenada. La motivación principal detrás de esta métrica parte de que dos imágenes pueden ser muy similares a nivel perceptual para un observador humano incluso si sus valores de píxel no coinciden perfectamente, o por el contrario, su comparación puede tener un error numérico relativamente bajo y aun así presentar diferencias visuales importantes. [(Zhang et al., 2018)]

El procedimiento general para calcular LPIPS consiste en ingresar la imagen original y la imagen reconstruida a una red convolucional preentrenada, como AlexNet, VGG o SqueezeNet, y comparar las activaciones internas que generan en distintas capas de dicha red. Estas activaciones son normalizadas y posteriormente la importancia de cada una es ajustada de acuerdo a pesos aprendidos a partir de juicios humanos de similitud perceptual. Dichos pesos fueron obtenidos por Zhang et al. durante el entrenamiento original de LPIPS, en el cual se pidió a distintas personas elegir cuál de dos parches distorsionados era más similar a un parche de referencia dentro del conjunto de datos BAPPS. De esta forma, la métrica busca aproximar mejor la percepción humana que una comparación directa en el espacio de píxeles. [(Zhang et al., 2018), (Zhang, s.f.)]

De forma simplificada, LPIPS puede expresarse como:

$$d(x, \hat{x}) = \sum_{l} \frac{1}{H_l W_l} \sum_{h,w} \left\| w_l \odot \left(\hat{y}^l_{hw} - y^l_{hw} \right) \right\|_2^2$$

Donde:

- x representa la imagen original y $\hat{x}$ la imagen reconstruida o generada.
- y^l_{hw} y $\hat{y}^l_{hw}$ representan las activaciones normalizadas de ambas imá-

genes en la capa l de la red neuronal, para la posición espacial (h, w).

- H_l y W_l representan las dimensiones espaciales del mapa de características en la capa l.
- w_l representa los pesos de calibración aprendidos a partir de juicios humanos de similitud perceptual durante el entrenamiento de LPIPS.
- $\odot$ indica una multiplicación elemento a elemento.

Para esta métrica un valor menor indica una mayor similitud perceptual entre la imagen original y la imagen reconstruida, mientras que un valor mayor indica una diferencia perceptual más grande, siendo 0 una similitud perfecta. Debido a que LPIPS opera sobre activaciones profundas, resulta particularmente útil para evaluar diferencias de textura, estructura y apariencia que pueden no ser representadas adecuadamente por métricas basadas en píxeles. [(Zhang et al., 2018), (Zhang, s.f.)]

2.9.7 Funciones de pérdida para inpainting

En el entrenamiento de redes neuronales, la función de pérdida define el error que se busca optimizar durante la retropropagación. Aunque una métrica de calidad puede utilizarse después del entrenamiento para comparar resultados con otros modelos, una función de pérdida participa directamente en la actualización de los pesos internos del modelo. Por este motivo, en restauración de imágenes la función de pérdida elegida no solo afecta el valor numérico del error, sino también el tipo de reconstrucción que el modelo aprende a favorecer.

Las pérdidas píxel a píxel, como L1 y L2, comparan directamente la imagen reconstruida contra la imagen objetivo en el espacio de píxeles. L2 penaliza de forma más agresiva los errores grandes debido a su término cuadrático, mientras que L1 mide las diferencias absolutas y suele ser menos sensible a

valores extremos. Estas funciones de pérdida son relativamente fáciles de optimizar y proporcionan una señal de entrenamiento estable, pero pueden favorecer reconstrucciones visualmente suaves o difuminadas cuando varios resultados plausibles presentan valores de píxel diferentes.

En tareas de restauración de imagen, los resultados cuantitativos no siempre coinciden con la calidad perceptual cualitativa de la reconstrucción. Dos imágenes pueden tener errores de píxel relativamente bajos y aun así diferir en estructura, textura o contraste. Por este motivo, algunos trabajos incorporan pérdidas estructurales o perceptuales basadas en métricas como SSIM o MS-SSIM. Mientras SSIM evalúa la similitud local considerando luminancia, contraste y estructura, MS-SSIM extiende esta comparación a múltiples escalas de la imagen para capturar información estructural en diferentes resoluciones. [(Zhao et al., 2017)]

Zhao et al. propusieron Mix Loss como una función de pérdida diseñada para restauración de imágenes que combina la fidelidad de L1 con la consistencia estructural de MS-SSIM. La intención de esta combinación es conservar una señal de error píxel a píxel útil para el entrenamiento, sin ignorar la similitud perceptual de la reconstrucción. De esta forma, Mix Loss representa un punto intermedio entre pérdidas puramente numéricas y pérdidas orientadas a preservar estructura visual. [(Zhao et al., 2017)]

CAPÍTULO 3: DESARROLLO TEÓRICO EXPERIMENTAL, RESULTADOS Y DISCUSIÓN

3.1 Equipo

El desarrollo experimental del presente proyecto se llevó a cabo en dos estaciones de trabajo de manera simultánea. Ya que el entrenamiento de modelos de redes neuronales es computacionalmente exigente, incluso para los modelos más sencillos se hizo uso de equipos con procesadores, memoria RAM y tarjetas gráficas de consumidor de gama media-alta, a continuación se detallan las especificaciones técnicas de cada una:

3.1.1 Estación de trabajo 1

- **Procesador:** Intel Core i7 de 11ª generación, 2.8 GHz
- **Memoria RAM:** 16 GB DDR5
- **Tarjeta gráfica:** NVIDIA GeForce GTX 1650 con soporte CUDA
- **Sistema operativo:** Windows 11 Pro x86_64
- **Entorno de desarrollo:** Visual Studio Code
- **Dependencias:** Control de paquetes y entorno virtual configurado mediante Miniconda

3.1.2 Estación de trabajo 2

- **Procesador:** Intel Core i7 de 14ª generación, 3.4 GHz

- **Memoria RAM:** 32 GB DDR5
- **Tarjeta gráfica:** NVIDIA GeForce RTX 4070 Ti Super con soporte CUDA
- **Sistema operativo:** Windows 11 Pro x86_64
- **Entorno de desarrollo:** Visual Studio Code
- **Dependencias:** Control de paquetes mediante y entorno virtual configurado mediante Miniconda

3.2 Materiales: lenguaje de programación, librerías

El entorno computacional se construyó con Python 3.9 y PyTorch con soporte de aceleración por GPU mediante CUDA. La siguiente tabla resume las bibliotecas principales utilizadas directamente por los flujos de entrenamiento, inferencia, visualización, evaluación, registro de experimentos y sincronización de puntos de control (checkpoints). Para mejorar la legibilidad, se omitieron de la tabla principal las bibliotecas de bajo nivel y las dependencias secundarias. Sin embargo, el archivo completo del ambiente conda podrá ser encontrado en la sección de anexos.

Tabla 1: Dependencias principales utilizadas en el flujo de trabajo de reconstrucción de imágenes con CNN.

Paquete	**Versión**	**Función en el proyecto**
`Python`	3.9.21	Lenguaje principal utilizado para los programas (scripts) de entrenamiento, evaluación, gestión de puntos de control, registro de experimentos y manejo de datos.

Paquete	Versión	Función en el proyecto
`PyTorch`	2.5.1	Marco de trabajo principal de deep learning. Se utiliza para definir los modelos de autocodificador convolucional, U-Net y CNN, realizar operaciones tensoriales, entrenar los modelos mediante diferenciación automática y ejecutar evaluaciones aceleradas por GPU.
`pytorch-cuda`	12.1	Soporte de entorno de ejecución (runtime) CUDA para la ejecución de PyTorch en GPU. La compilación instalada de PyTorch utiliza CUDA 12.1 y soporte cuDNN 9, lo que permite operaciones tensoriales en GPU y entrenamiento con precisión mixta (mixed precision). Esta dependencia es vital para las operaciones de convolución de las CNN entrenadas.
`torchvision`	0.20.1	Proporciona transformaciones de imagen para el preprocesamiento, la interfaz de la base de datos CelebA, utilidades de carga de datos compatibles con PyTorch y modelos de visión preentrenados utilizados durante la evaluación y la extracción de características perceptuales.

Paquete	Versión	Función en el proyecto
`NumPy`	2.0.1	Se utiliza para operaciones numéricas fuera de la red neuronal, incluyendo generación de máscaras, estadísticas de evaluación, muestreo aleatorio y cálculos de media y covarianza relacionados con la métrica de distancia de Fréchet de Inception.
`SciPy`	1.13.1	Se utiliza en la evaluación para calcular raíces cuadradas matriciales requeridas por la métrica de distancia de Fréchet de Inception.
`Pillow`	11.1.0	Se utiliza indirectamente en rutinas de preprocesamiento de imágenes y generación de máscaras, especialmente para convertir y redimensionar patrones de máscara.
`Matplotlib`	3.9.4	Se utiliza para generar visualizaciones de reconstrucción, figuras de evaluación y tablas gráficas de comparación entre distintos modelos.
`pytorch_msssim`	1.0.0	Proporciona funciones de pérdida y métricas de similitud de imagen basadas en SSIM y MS-SSIM, utilizadas para optimizar y evaluar la calidad de reconstrucción de imagen de los modelos entrenados.
`AdaBelief PyTorch`	0.2.1	Proporciona acceso al optimizador AdaBelief utilizado en diversas configuraciones de entrenamiento.

Paquete	Versión	Función en el proyecto
`torch-optimizer`	0.3.0	Proporciona optimizadores adicionales, incluyendo RAdam, utilizados en la configuración de entrenamiento.
`tqdm`	4.67.1	Proporciona barras de progreso para la visualización en tiempo real de los ciclos de entrenamiento y evaluación.
`PyMongo`	4.11.3	Se utiliza para almacenar todos los datos relevantes de entrenamiento y evaluación en una base de datos noSQL MongoDB. Estos datos incluyen los metadatos de experimentos, hiperparámetros, configuraciones de modelos, resultados y tiempos de entrenamiento, métricas de evaluación y notas adicionales.
`rclone`	CLI externo; paquete Python 0.4.4 instalado	Se utiliza para sincronizar puntos de control con Google Drive. El repositorio invoca el ejecutable `rclone` mediante llamadas de subprocess para descargar y subir los archivos .pth de modelos. Así como las imágenes generadas para evaluar cualitativamente cada modelo.

3.2.1 Grupos Funcionales Principales

Las dependencias pueden organizarse en cuatro categorías funcionales.

1. En primer lugar, `PyTorch`, `torchvision` y `pytorch-cuda` conforman la base fundamental del entrenamiento y validación, proporcionando arquitecturas convolucionales, aceleración por GPU, precisión mixta,

manejo de conjuntos de imágenes y extractores de características preentrenados.

2. `NumPy`, `SciPy`, `pytorch_msssim` y `Matplotlib` sirven como apoyo a la evaluación cuantitativa y cualitativa, incluyendo la función de pérdida SSIM/MS-SSIM, el cálculo de operaciones y estadísticas relacionadas con FID, generación de figuras e imágenes para la inspección visual de resultados y tablas de comparación entre modelos.

3. `AdaBelief PyTorch` y `torch-optimizer` proporcionan métodos de optimización complejos más allá de los optimizadores estándar de PyTorch.

4. Finalmente, `PyMongo` y `rclone` permiten que exista una persistencia de experimentos en la nube y la sincronización de puntos de control, registros y visualizaciones entre almacenamiento local, MongoDB y Google Drive.

3.2.2 Dependencias Excluidas de la Tabla Principal

Algunos paquetes presentes en el entorno virtual conda no se incluyeron en la tabla principal porque corresponden a dependencias transitivas (aquellos paquetes que las dependencias directas necesitan para funcionar) o bibliotecas de soporte en tiempo de ejecución.

Algunos ejemplos son los componentes de bajo nivel de CUDA, las bibliotecas MKL/BLAS, utilidades de certificados y HTTP, bibliotecas auxiliares de la API de Google, `pandas`, `tabulate`, `gdown`, `torch-fidelity` y utilidades de compresión de archivos. Estos paquetes pueden ser necesarios para reproducir de manera fiel el comportamiento del entorno virtual utilizado, pero no son particularmente relevantes para describir la metodología del flujo de trabajo de entrenamiento y evaluación.

3.3 Metodología experimental y diseño iterativo

Debido a la naturaleza experimental e iterativa del entrenamiento de modelos de redes neuronales, el desarrollo metodológico de este proyecto no se llevó a cabo como una secuencia completamente lineal, sino como un proceso de diseño, entrenamiento, evaluación y ajuste progresivo. Por este motivo, en esta sección se describen tanto los componentes metodológicos utilizados para construir el flujo de trabajo experimental como los resultados parciales que justificaron las decisiones tomadas en cada etapa del desarrollo y evolución del modelo. De esta forma resulta más sencillo seguir la secuencia lógica de cómo cada modificación en la arquitectura, los hiperparámetros, las funciones de pérdida y los criterios de evaluación contribuyó a la configuración final del modelo propuesto.

3.3.1 Creación del conjunto de datos para entrenamiento

Se partió de la base de datos existente de rostros humanos CelebA la cual cuenta con 202,599 imágenes de tamaño 178x218 píxeles. Fue aplicada la técnica de aumento de datos de manera dinámica explorando las diferentes opciones de enmascaramiento aleatorio y adición de ruido disponibles en PyTorch para generar así un conjunto de datos de pares de imágenes con la imagen modificada y su respectiva imagen original.

En esta sección se llevó a cabo el desarrollo de un programa en lenguaje de programación Python capaz de generar dinámicamente 3 tipos de enmascaramiento de imágenes:

1. Ausencia completamente aleatoria (Missing Completely at Random, MCAR): Dada cierta variable “p” existe para cada píxel independiente una probabilidad de que este sea o no enmascarado, lo cual da como resultado una máscara de píxeles aleatorios.

2. Rectangle Generator: Para cada imagen se generará una máscara que consista en un rectángulo cuya posición y tamaño serán aleatorios, sin embargo, el área total de dicho rectángulo se encuentra acotada dentro de un mínimo y un máximo de porcentaje del área total de la imagen.

3. Random Pattern: Genera máscaras con patrones irregulares similares a masas irregulares (blobs), al igual que Rectangle Generator la posición y tamaño de estas masas es aleatoria y el área que ocupan entre todas estará acotada por un máximo porcentaje de acuerdo con el área total de la imagen. Sin embargo, a diferencia de Rectangle Pattern donde únicamente se genera 1 rectángulo, Random Pattern es capaz de generar varias masas irregulares de forma discontinua cada una con una forma y tamaño diferente. La implementación de esta máscara está inspirada por el trabajo de Pathak et al. (2016) y su implementación puede entenderse de la siguiente forma:

    a. Primero se genera una matriz de valores aleatorios entre 0 y 1 en baja resolución.

    b. Esta matriz se interpola bicúbicamente para escalarla a una resolución muy grande, lo que permite formar manchas suaves e irregulares.

    c. Luego, estos valores se binarizan aplicando un umbral de densidad: los valores por debajo del umbral se consideran enmascarados (1), y los demás visibles (0).

    d. A partir de esa máscara gigante, se generan las máscaras individuales del lote: para cada imagen se toma un parche aleatorio del tamaño correspondiente y se valida que tenga una proporción adecuada de píxeles enmascarados (según la densidad deseada).

    e. Si el parche cumple con la cobertura esperada, se acepta como máscara. Si no, se intenta con otra posición aleatoria. De esta

forma se asegura que cada imagen tenga una máscara diferente pero coherente con el patrón general.

Adicionalmente se definió el tipo de máscara Mixture como una combinación de los otros 3 anteriormente mencionados, dividiendo la base de datos en partes iguales de forma aleatoria y aplicando uno de los tipos de máscara a cada segmento. En el entrenamiento de nuestros modelos se optó por siempre utilizar enmascaramiento tipo Mixture para evitar el problema de sobreajuste y mejorar la robustez y generalización de los modelos.

En cuanto a la aplicación de las máscaras a las imágenes de la base de datos CELEB-A, esta se llevó a cabo por lotes, directamente a sus representaciones como tensores y fue implementada en 2 modalidades:

1. Como inpainting ciego: tras generarse las máscaras se expanden sus dimensiones para cubrir los 3 canales RGB y se reemplazan directamente los valores de las regiones cubiertas por la máscara, dando como resultado final una imagen enmascarada en 3 canales que puede visualizarse de forma sencilla usando Matplotlib.
2. Como inpainting no ciego: Al igual que en el caso del inpainting ciego se expanden las dimensiones de la máscara y se reemplazan directamente los valores de las regiones enmascaradas. Sin embargo, a diferencia del caso ciego, en esta ocasión se genera una copia binaria en 1 canal la cual le indicará al modelo la posición de los píxeles enmascarados. De esta forma el resultado final en esta modalidad es la imagen enmascarada en 3 canales y la máscara binaria en 1, dándonos un total de 4 canales.

Es importante mencionar que tanto nuestros antecedentes como nuestra experimentación empírica nos han llevado a la conclusión de que tanto para el caso del inpainting ciego como no ciego, los valores con los cuales se reemplazan los píxeles enmascarados en los canales RGB de la imagen

son importantes y afectan el desempeño del modelo. De acuerdo con el trabajo de (Liu et al., 2018) reemplazar los píxeles enmascarados por un valor RGB correspondiente a 127.5 ([0.5, 0.5, 0.5] para una normalización en el intervalo [0, 1]) en todos los canales produce peores resultados en general que reemplazar dichos píxeles con el valor de la media RGB de la base de datos ImageNet ([0.485, 0.456, 0.406] para una normalización en el intervalo [0, 1]). Estos valores le facilitan al modelo identificar y reemplazar los píxeles enmascarados lo cual se traduce en una disminución significativa en los errores de entrenamiento y validación.

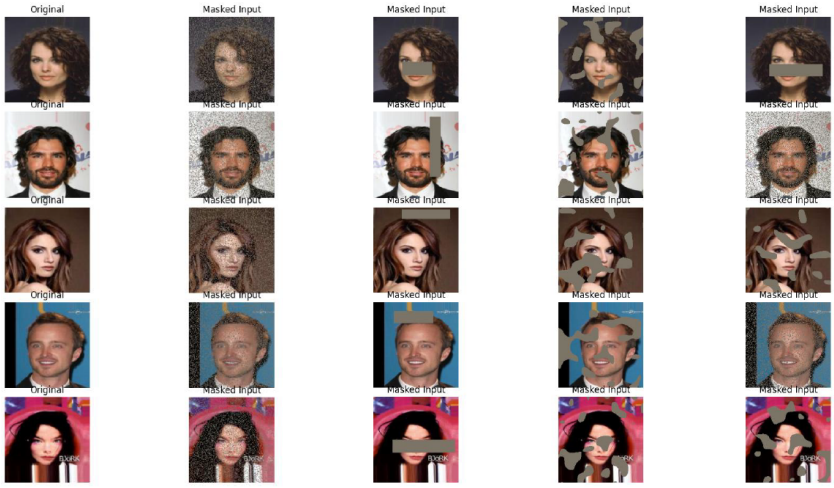


Figura 1: De izquierda a derecha: Imagen original, máscara MCAR, máscara Rectangle Generator, máscara Pattern Generator, máscara Mixture.

3.3.2 Creación de un flujo de trabajo de entrenamiento

Fue creado un flujo de trabajo de entrenamiento semiautomático en lenguaje de programación Python para facilitar una rápida iteración entre diferentes arquitecturas y variantes de modelos de redes neuronales artificiales. Este flujo de trabajo se desarrolló con el objetivo de ser capaz de precargar dinámicamente los datos generados en la sección 3.3.1, tener la funcionalidad de definir y seleccionar modelos e hiperparámetros,

guardar un registro detallado de la arquitectura en una base de datos NoSQL, así como guardar los pesos y visualizaciones del mejor modelo de cada tipo en un depósito de almacenamiento para poder reproducir los experimentos de ser necesario.

En esta sección se llevó a cabo el desarrollo de una herramienta informática (software tool) en lenguaje de programación Python capaz de manejar dinámicamente todas las configuraciones esenciales para el entrenamiento de modelos de redes neuronales artificiales enfocados a la tarea de inpainting facial, dicho sistema consiste a grandes rasgos de los siguientes módulos funcionales:

1. **Módulo de carga de datos (Dataloader Module):** Se encarga de descargar la base de datos CELEB-A en el entorno local, divide la base de datos en conjuntos (sets) de entrenamiento y validación de acuerdo con cómo estos se encuentran definidos por los autores originales de dicha base de datos (162771 imágenes de entrenamiento y 19961 imágenes de validación respectivamente). Asimismo, realiza el siguiente preprocesamiento para todas las imágenes:
   (a) Ajusta el tamaño de la imagen a aquel que sea definido dentro de los hiperparámetros del flujo de trabajo.
   (b) Convierte cada imagen a formato tensor de pytorch para que su información pueda ser procesada mediante GPU.
   (c) Normaliza los valores RGB de la imagen al rango definido dentro de los hiperparámetros del flujo de trabajo ([-1 1] o [0 1]).
2. **Módulo generador de máscaras (Mask Generator):** Como se detalla en la etapa 2, este módulo se encarga de generar y aplicar el enmascaramiento dinámico a las imágenes de los cargadores de datos en las modalidades de inpainting ciego y no ciego. Es importante destacar que la configuración de este módulo permite elegir no solamente las máscaras tipo mixture con una probabilidad equilibrada para

los 3 tipos de enmascaramiento, sino que además permite al usuario definir otros parámetros como el área proporcional que cubrirán las máscaras, el tamaño mínimo y máximo de los parches, así como la probabilidad de que cada tipo de máscara se presente dentro del lote (Ejemplo: el enmascaramiento podría configurarse para tener una probabilidad del 70% para Pattern Generator y 30% para MCAR).

3. **Módulo selector de modelo (Model Selector Module):** Se encarga de mapear los diferentes tipos y variantes de modelos de redes neuronales (definidos por el usuario en el archivo models.py como clases) a diferentes palabras clave que pueden ser utilizadas dentro de los hiperparámetros para seleccionar el modelo a entrenar, sin la necesidad de modificar directamente el flujo general de entrenamiento.

4. **Módulo optimizador y escalador (Optimizer Module):** Se encarga de mapear los diferentes tipos de optimizadores de función de pérdida que ofrece Pytorch (Adam, AdamW, RAdam, Adabelief, etc) a palabras clave que pueden ser utilizadas dentro de los hiperparámetros del sistema, asimismo transmite los parámetros importantes (Decaimiento de pesos (weight decay), Tasa de aprendizaje (learning rate), BETAS) al optimizador seleccionado. El uso de un optimizador adecuado modifica el comportamiento de la regla de actualización de pesos (weight update rule) y mejora significativamente la convergencia del modelo. Su otra función es definir si se utilizará o no el escalado dinámico de gradientes (GradScaler) durante el entrenamiento, una técnica que permite mitigar problemas numéricos asociados al uso de precisión reducida (como float16). El escalado dinámico ajusta automáticamente la magnitud de los gradientes para evitar que valores muy pequeños se redondeen a cero debido a la limitada precisión, lo cual ayuda a mantener la estabilidad del entrenamiento y aprovechar mejor el rendimiento computacional de los dispositivos con soporte para precisión mixta (como las GPUs de NVIDIA con Tensor Cores). Nota: El entrenamiento con precisión

mixta implica usar valores tipo float16 para cálculos intermedios y float32 para acumulaciones de gradiente, GradScaler facilita esto sin perder estabilidad.

5. **Módulo selector de planificador (Scheduler Selector Module):** Se encarga de mapear los diferentes tipos de planificador de tasa de aprendizaje (learning rate scheduler, una herramienta que permite ajustar el valor de la tasa de aprendizaje dinámicamente a lo largo del proceso de entrenamiento) que ofrece Pytorch (StepLR, ReduceLROnPlateau, CosineAnnealingLR, etc) a palabras clave que pueden ser utilizadas dentro de los hiperparámetros del sistema, asimismo transmite los parámetros importantes asociados a cada tipo de planificador en forma de diccionario. El uso de planificadores de tasa de aprendizaje en el entrenamiento de un modelo ayuda a mitigar el problema de sobreajuste y mejora la convergencia del modelo en sus últimas épocas.

6. **Módulo selector de función de pérdida (Criterion Selector Module):** Se encarga de mapear los diferentes tipos y variantes de funciones de pérdida (definidas en el archivo losses.py como clases) a diferentes palabras clave que pueden ser utilizadas dentro de los hiperparámetros para seleccionar la función de pérdida de la capa de salida del modelo a entrenar, sin la necesidad de modificar directamente el flujo general de entrenamiento. Estas funciones de pérdida pueden ser importadas directamente de la librería estándar de Pytorch (pérdida L1, pérdida L2), librerías de terceros (SSIM, MS-SSIM) o ser definidas directamente por el usuario como es el caso de Mix Loss, una función de pérdida que combina la pérdida L1 y MS-SSIM y se basa en el trabajo de (Zhao et al., 2017).

7. **Módulo selector de inicializador de pesos (Initializer Selector Module):** Se encarga de mapear los diferentes tipos de inicializadores de pesos que ofrece Pytorch (Xavier Uniform, Xavier Normal, Kaiming Uniform, Kaiming Normal, Orthogonal, etc) a palabras clave que

pueden ser utilizadas dentro de los hiperparámetros del sistema, el uso de inicializadores de pesos ayuda a reducir los errores de pérdida y validación, así como contribuir a que la convergencia del modelo se realice en menos épocas.

8. **Módulo de parada temprana (Early Stopping Module):** Se encarga de monitorear época por época la convergencia de entrenamiento del experimento en ejecución. En caso de que el error de validación se quede estancado o empeore durante varias épocas consecutivas, el mecanismo de parada temprana se encargará de abortar el experimento en curso, guardar los mejores pesos obtenidos y agregar una nota al registro de dicho experimento para indicar que el entrenamiento fue inestable.

9. **Módulo de Registros (Logger Module):** Se encarga de extraer información relevante y detallada del resto de los módulos, así como de los resultados de entrenamiento y validación del modelo época por época. Estos registros se guardan en formato JSON y son exportados en tiempo real a una base de datos NoSQL de MongoDB, de esta forma podemos acceder a todos los detalles relevantes de cada experimento realizado para llevar a cabo las comparaciones pertinentes entre los diferentes modelos y variantes que se vayan entrenando. Siendo MongoDB un sistema no relacional basado en “documentos” es posible agregar o eliminar campos según lo requiera el experimento y la arquitectura, esta flexibilidad a su vez no sacrifica la capacidad de consultar información por campos lo cual convierte a este módulo en una pieza esencial del flujo de trabajo.

10. **Módulo de guardado (Checkpoint Manager Module):** Se encarga de guardar los pesos y los sesgos de cada variante de modelo generada en formato pth. Para asegurarse de no consumir demasiado espacio de almacenamiento, este módulo realiza comparaciones época por época con los resultados del mejor modelo global para cada tipo

de experimento dado, de esta forma el sistema se asegura de únicamente guardar el mejor modelo de cada tipo. Una vez guardados los modelos en almacenamiento local se hace uso de la librería rdrive para subir los archivos pth a un depósito de Google Drive. Este módulo además cuenta con métodos para comparar y validar la integridad de los archivos guardados en almacenamiento local con aquellos almacenados en la nube y en caso necesario es capaz de descargar todos los modelos a un dispositivo nuevo.

11. **Módulo de Visualización de Resultados:** Se encarga de generar una figura comparativa a partir del cargador de datos de validación en donde se muestran 5 ejemplos de imágenes en 3 columnas: a) la imagen original b) la imagen enmascarada c) la imagen reconstruida por el modelo. Esta figura se muestra brevemente al usuario al final de cada época y se guarda en un archivo png mediante la biblioteca matplotlib para poder observar la convergencia del modelo época por época de forma cualitativa. Posteriormente las visualizaciones de la inferencia realizada por el modelo con el mejor desempeño de cada tipo se suben a la nube haciendo uso del módulo de guardado.

12. **Archivo de hiperparámetros (hyperparameters.py):** En este archivo se definen todas las variables globales que controlan la configuración y comportamiento de todos los demás módulos que afectan al ciclo principal de entrenamiento. Además de los hiperparámetros mencionados anteriormente también se encuentran en este archivo: el tamaño de lote (batch size), el número de hilos del CPU, el número de épocas por experimento, el nombre descriptivo del tipo de experimento, y una variable booleana que le indica al sistema si se va a llevar a cabo inpainting ciego o no ciego.

13. **Archivo de ejecución principal (main.py):** Este archivo se encarga de llamar secuencialmente a todos los subprocesos relevantes para realizar de forma automática el entrenamiento y evaluación de un tipo de experimento predefinido; primero se descargan del depósito de

Google Drive los modelos y visualizaciones al almacenamiento local en caso de no existir, a continuación, se ejecuta el ciclo de entrenamiento principal una cantidad de veces definida por el usuario (Ej. 30 lanzadas para un tipo de experimento), cada cierto número de lanzadas se suben los nuevos modelos y visualizaciones generados a Google Drive y finalmente se lleva a cabo una copia de seguridad local de la base de datos NoSQL de MongoDB.

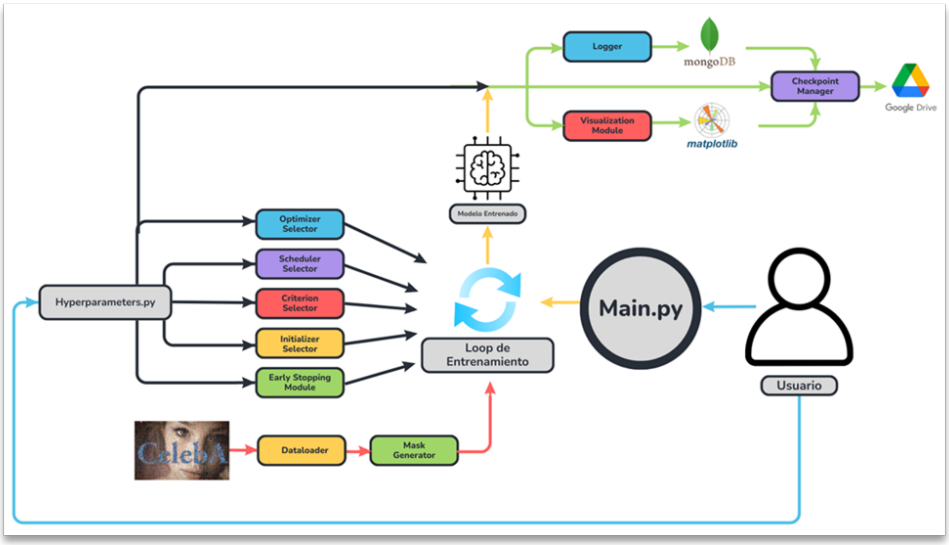


Figura 2: Flujo de trabajo de entrenamiento desarrollado.

3.3.3 Diseño y resultados de la iteración de entrenamiento de un autocodificador de eliminación de ruido

En esta sección se llevó a cabo el entrenamiento de varios modelos de autocodificadores de eliminación de ruido, variando a su vez sus respectivos hiperparámetros como lo son su topología, funciones de transferencia, funciones de error, funciones de normalización, inicialización de pesos y optimización, número de épocas de entrenamiento, tamaño de lote de muestras de entrenamiento, entre otros.

Durante esta etapa se tomaron en cuenta dos criterios principales para determinar si una iteración del modelo representaba una mejora respecto al modelo de referencia: el valor de la función de pérdida en el conjunto de validación y la calidad visual de las reconstrucciones. Cuando se identificaba una mejora, dicha iteración se convertía en el nuevo punto de referencia para comparaciones posteriores. El error de entrenamiento se utilizó como referencia de la convergencia, pero no fue suficiente por sí solo para seleccionar un modelo. Al comparar experimentos con la misma función de pérdida se buscaron menores errores de validación sin empeorar los resultados cualitativos. Cuando hubo cambios o ajustes en la función de pérdida, sus valores no se compararon directamente entre sí y se recurrió a la inspección visual para valorar las reconstrucciones.

Este proceso iterativo se detalla a continuación para los primeros 400 experimentos:

1. **Definición del modelo base (Experimentos 1-30):** Para realizar una comparación exhaustiva de arquitecturas e hiperparámetros es necesario tener un grupo de control con el cual comparar el desempeño; es por esto que se llevaron a cabo 30 experimentos con un modelo base de autocodificador convolucional de eliminación de ruido no optimizado, para la definición de dicho modelo se eligieron los siguientes parámetros:

- Codificador: Conformado por 4 bloques convolucionales cada uno con los siguientes submódulos:
  - Capa Convolucional (Conv2d):
    - Tamaño de núcleo (kernel size): 3x3
    - Paso (stride) 2
    - Relleno (padding) 1
    - Canales de entrada y salida (input/output channels):
      - Codificador 1: 3 IN, 32 OUT
      - Codificador 2: 32 IN 64 OUT
      - Codificador 3: 64 IN 128 OUT
      - Codificador 4: 128 IN 256 OUT
  - Normalización por Lote (batch normalization; BatchNorm2d)
  - Función de activación Tanh
  - Dropout:
    - Probabilidad 0.3
- Decodificador: Conformado por 4 bloques deconvolucionales cada uno (con la excepción del decodificador 4) con los siguientes submódulos:
  - Capa Convolucional Transpuesta (ConvTranspose2d)
    - Tamaño de núcleo (3x3)
    - Paso 2
    - Relleno 1
    - Canales de entrada y salida:
      - Decodificador 1: 256 IN, 128 OUT
      - Decodificador 2: 128 IN 64 OUT
      - Decodificador 3: 64 IN 32 OUT
      - Decodificador 4: 32 IN 3 OUT

  - Normalización por Lote (BatchNorm2d)
  - Función de activación Tanh
  - Dropout:
    - Probabilidad 0.3
  - Para la capa de salida (Decodificador 4) únicamente se utilizan:
    - Capa Convolucional Transpuesta
    - Función de activación Tanh
- Preprocesamiento de imágenes:
  - Ajuste de tamaño (128x128)
  - Conversión a tensores pytorch
  - Normalización [-1 1]
- Hiperparámetros:
  - Número de épocas: 15
  - Tamaño de lote: 128
  - Tasa de aprendizaje: 0.0003
  - Probabilidad de Dropout: 0.3
  - Optimizador: Adam
    - β_1 0.5 β_2 0.999
  - Inicialización de pesos: Aleatoria
  - Enmascaramiento: Mixture
  - Escalado dinámico de gradientes: Activado
  - Función de pérdida SSIM + L1 Global (80% L1, 20% SSIM)
  - Planificador: StepLR
    - Step_size: 5
    - Gamma: 0.5
- Resultados:

  - Tiempo promedio de entrenamiento: 1707.5 segundos
  - Error mínimo de entrenamiento: 0.2311302191407988 (id 30)
  - Error mínimo de validación: 0.16421574612076467 (id 23)
- Notas Adicionales: Los resultados cualitativos generaban artefactos tipo checkerboard y la calidad general de la reconstrucción era mala.

2. **Barrido de Probabilidad de Dropout (Experimentos 31-50):** La desactivación aleatoria de neuronas (Dropout) es una técnica de regularización muy comúnmente utilizada en redes neuronales para evitar el problema de sobreajuste y a su vez mejorar el error de validación en entrenamientos largos y la capacidad de generalización del modelo a costa de un mayor error de entrenamiento. Consiste en desactivar secciones de la red de forma aleatoria a lo largo del proceso de entrenamiento, esto fuerza al sistema a no depender tan fuertemente de secciones determinadas de su arquitectura durante la inferencia.

   Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la siguiente tabla. En este caso el Dropout 0.1 fue considerado como la alternativa más óptima debido a que proporciona una tasa de error significativamente más baja que el modelo base sin caer en el problema de sobreajuste, como es el caso de Dropout 0; donde el error mínimo de entrenamiento y el error mínimo de validación son casi iguales y podrían cruzarse al entrenar el modelo durante un número de épocas mayor.

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de incidentes de parada temprana	Número de ejecuciones
Dropout 0	0.15568368080651984 (id 31)	0.14582100329108727 (id 34)	0	5
Dropout 0.1	0.19743729888830544 (id 37)	0.1517769877727215 (id 37)	1	5
Dropout 0.2	0.21593416444722963 (id 42)	0.1576648743297809 (id 42)	1	5
Dropout 0.3	0.2311302191407988 (id 30)	0.16421574612076467 (id 23)	2	Modelo base
Dropout 0.4	0.2445437776381677 (id 48)	0.17708413895124045 (id 48)	3	5

Tabla 2: Resultados cuantitativos del barrido de probabilidad de Dropout.

3. **Barrido de Tasa de aprendizaje (Experimentos 51-65):** La tasa de aprendizaje representa uno de los hiperparámetros más importantes de cualquier modelo de redes neuronales artificiales, determina en gran medida el comportamiento de la regla de actualización de pesos y su valor puede ser el factor clave que determine que un modelo converja o no. Su papel se representa en la siguiente ecuación $w_{new} = w_{old} - lr \left(\frac{\partial Error}{\partial w_{old}} \right)$ Donde w_{new} representa el nuevo valor del peso después de cada retropropagación y $\left(\frac{\partial Error}{\partial w_{old}} \right)$ representa a la derivada parcial del error de salida con respecto al peso a actualizar, esta operación se realiza para todos los pesos de la red. Un valor demasiado pequeño puede provocar que la red neuronal demore demasiado tiempo en converger o se atasque en mínimos locales con facilidad, mientras que un valor demasiado grande puede provocar que el error se estanque en un intervalo y sea incapaz de dar los pasos finos necesarios para llegar al mínimo global.

Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la tabla 2. En este caso la tasa de aprendizaje con valor de 0.0005 fue considerada como la alternativa más óptima debido a que fue capaz de superar al modelo base consistentemente tanto en error de entrenamiento como en error de validación.

Tipo de Experimento	**Error mínimo de entrenamiento**	**Error mínimo de validación**	**Número de ejecuciones**
LR 0.0005	0.22749197073046518 (id 53)	0.16397495290789849 (id 52)	5
LR 0.001	0.2254817932609867 (id 60)	0.1647396128720198 (id 57)	5
LR 0.0001	0.24572241462111097 (id 64)	0.1800487247797159 (id 62)	5
LR 0.0003	0.2311302191407988 (id 30)	0.16421574612076467 (id 23)	Modelo base

Tabla 3: Resultados cuantitativos del barrido de tasa de aprendizaje.

4. **Barrido de planificador (Experimentos 66-80):** El planificador de tasa de aprendizaje es una herramienta que permite ajustar el valor de la tasa de aprendizaje dinámicamente a lo largo del proceso de entrenamiento, la idea es que conforme avance el proceso de entrenamiento del modelo se reduzca el valor de la tasa de aprendizaje para evitar un sobrepaso (overshoot; saltarse el mínimo global al no poder realizar pasos suficientemente finos en las etapas más avanzadas del entrenamiento).

   Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la tabla 3. En este caso el planificador de tipo Reduce on Plateau con un umbral de 0 fue considerado como la alternativa más óptima debido a que fue capaz de superar al modelo base consistentemente tanto en error de entrenamiento como en error de validación y no presentó errores de estabilidad en su ejecución como el caso de umbral 0.002.

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de ejecuciones	Número de incidentes de parada temprana
Planificador Cosine Annealing LR	0.23239219046941717 (id 69)	0.16748123740156493 (id 68)	5	0
Planificador Reduce on Plateau (umbral 0.002)	0.2277151583781782 (id 73)	0.16080461060389495 (id 73)	5	4
Planificador Reduce on Plateau (umbral 0)	0.22933865714607374 (id 78)	0.16126450179861143 (id 78)	5	0
Planificador StepLR	0.2311302191407988 (id 30)	0.16421574612076467 (id 23)	Modelo base	2

Tabla 4: Resultados cuantitativos del barrido del planificador de tasa de aprendizaje.

5. **Barrido de Optimizador (Experimentos 81-95):** El entrenamiento de toda red neuronal artificial implica una exploración del espacio de soluciones del problema a tratar. En este contexto, los optimizadores se presentan como herramientas esenciales que permiten recorrer dicho espacio de forma más eficiente En términos generales, el optimizador de una red neuronal artificial es un algoritmo que actúa sobre la regla de retropropagación ajustando los pesos del modelo con base en los gradientes previamente calculados, lo cual le permite al modelo aprender patrones útiles. Su objetivo es encontrar un mínimo global o aceptable respecto a la función de pérdida.

   Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la tabla 4. En este caso el optimizador de tipo Adabelief fue considerado como la alternativa más óptima debido a que fue capaz de superar al modelo base tanto en error de

entrenamiento como en error de validación, a pesar de que la diferencia en error de validación fue pequeña, el hecho de haber podido llegar a un mejor mínimo en 1/6 de las ejecuciones y sin incrementar el tiempo promedio de cómputo sustancialmente es suficiente para justificar esta decisión.

Tipo de Experimento	**Error mínimo de entrenamiento**	**Error mínimo de validación**	**Número de ejecuciones**
AdamW	0.2324181868103128 (id 85)	0.16626513682496855 (id 85)	5
RAdam	0.23347225570875518 (id 88)	0.16717978748373497 (id 88)	5
Adabelief	0.22877889228457549 (id 93)	0.1642135962461814 (id 94)	5
Adam	0.2311302191407988 (id 30)	0.16421574612076467 (id 23)	Modelo base

Tabla 5: Resultados cuantitativos del barrido de optimizadores.

6. **Barrido de BETAS (Experimentos 96-115):** Los parámetros conocidos como BETAS (β_1 y β_2) del optimizador Adam y sus variantes controlan la velocidad con la que se olvidan los gradientes pasados, mediante el uso de promedios móviles del gradiente (primer momento) y su cuadrado (segundo momento). β_1 regula el momento, ayudando a estabilizar la dirección del aprendizaje, mientras que β_2 ajusta el valor de la tasa de aprendizaje de forma adaptativa. Valores mayores implican que dichos promedios y su efecto en la retropropagación estarán más influenciados por gradientes anteriores, es por esto que valores adecuados de estos hiperparámetros contribuyen a una convergencia más rápida y estable.

   Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la tabla 5. En este caso los valores de BETAS β_1 0.9 β_2 0.95 fueron considerados como la alternativa más óptima debido a que fueron capaces de superar al modelo base tanto en error de entrenamiento como en error de validación.

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de ejecuciones
β_1 0.9 β_2 0.999	0.23048719998050785 (id 97)	0.16653330194262358 (id 97)	5
β_1 0.9 β_2 0.95	0.22556828504791424 (id 104)	0.15892468631649628 (id 105)	5
β_1 0.6 β_2 0.9	0.2256073405110986 (id 109)	0.15985622448034775 (id 110)	5
β_1 0.5 β_2 0.99	0.22944215297370962 (id 113)	0.16270422362364256 (id 113)	5
β_1 0.5 β_2 0.999	0.2311302191407988 (id 30)	0.16421574612076467 (id 23)	Modelo base

Tabla 6: Resultados cuantitativos del barrido de parámetros β_1 y β_2.

7. **Barrido de Inicializador de Pesos (Experimentos 116-140):** Los valores iniciales de los pesos de toda red neuronal determinan la posición inicial dentro del espacio de soluciones multidimensional desde donde el modelo parte en busca del error mínimo global. Es por esto que el papel de los inicializadores de pesos resulta crucial, ya que un inicializador optimizado para las funciones de transferencia y normalizaciones de valores empleadas en la arquitectura del sistema puede hacer una gran diferencia en cuanto al tiempo de convergencia. Los inicializadores de pesos, a diferencia de la inicialización de pesos aleatoria no controlada, ajustan los valores iniciales dentro de una distribución predefinida, lo cual tiende a producir resultados más estables y replicables.

   Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la tabla 6. En este caso el inicializador Xavier Normal fue considerado como la alternativa más óptima debido a que a pesar de presentar resultados ligeramente inferiores a la mejor iteración del modelo base, se mantiene consistentemente bastante cerca de este óptimo y presenta una desviación estándar mucho menor, lo cual implica que a la larga el entrenamiento será más estable y consistente.

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de ejecuciones	Desviación estándar de validación
Inicializador Xavier Uniform	0.23220660700316323 (id 119)	**0.16725039357940355** (id 118)	5	0.0039930631804174135
Inicializador Xavier Normal	0.23380154455607793 (id 123)	0.16708928232009596 (id 125)	5	0.002365147036790985
Inicializador Kaiming Uniform	0.2423398761699597 (id 129)	0.18630468577910692 (id 126)	5	0.0022857112673490823
Inicializador Kaiming Normal	0.24265493819220635 (id 131)	0.1846458736138466 (id 131)	5	0.005435042595351451
Inicializador Orthogonal	0.2344858701100702 (id 137)	0.16978807269762725 (id 137)	5	0.002241939989942036
Inicialización Aleatoria	0.2311302191407988 (id 30)	0.16421574612076467 (id 23)	Modelo base	0.006642241874407826

Tabla 7: Resultados cuantitativos del barrido de inicializadores de pesos

8. **Barrido de hiperparámetros óptimos y normalización (Experimentos 141-180):** Una vez obtenidos los hiperparámetros óptimos para esta arquitectura en particular mediante nuestro acercamiento basado en descenso por coordenadas (coordinate descent; una técnica de optimización de problemas multivariables que consiste en tratar la optimización del sistema de forma secuencial como si trabajáramos con varios problemas univariables) fijamos los valores óptimos y evaluamos la mejoría general en el desempeño del modelo. Asimismo se evalúa el efecto que tiene la normalización por lotes en el desempeño del modelo optimizado y el efecto que tiene cambiar la normalización inicial de las máscaras de un valor RGB 127 normalizado al valor de la media RGB de la base de datos ImageNet.

   Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la siguiente tabla. En este caso se

considera como la opción óptima el mantener la normalización por lotes (BatchNorm) e implementar la normalización de máscaras con la media de ImageNet. A pesar de que la desviación estándar aumenta con respecto al modelo base optimizado, los resultados cuantitativos del mejor caso son mejores, mientras que los resultados cualitativos (ver figura) muestran que normalizar a la media de ImageNet reduce la visibilidad de las máscaras a la salida. Esta noción se ve apoyada por la experimentación de (Liu et al., 2018).

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de ejecuciones	Desviación estándar de validación
Modelo Base sin optimizar (normalización por lote, máscaras RGB 127)	0.2311302191407988 (id 30)	0.16421574612076467 (id 23)	Modelo base	0.006642241874407826
Modelo base optimizado (normalización por lote, máscaras RGB 127)	0.18477270112566227 (id 141)	0.1395091611223343 (id 154)	15	0.002675677192504993
Modelo base optimizado (sin normalización por lote, máscaras RGB 127)	0.18483219351970925 (id 157)	0.1460346706593648 (id 157)	10	0.14152818272264822
Modelo base optimizado (normalización por lote, máscaras RGB ImageNet)	0.18477706463550622 (id 177)	0.13734915451361582 (id 171)	15	0.005857868851381317

Tabla 8: Resultados cuantitativos del barrido de normalización.

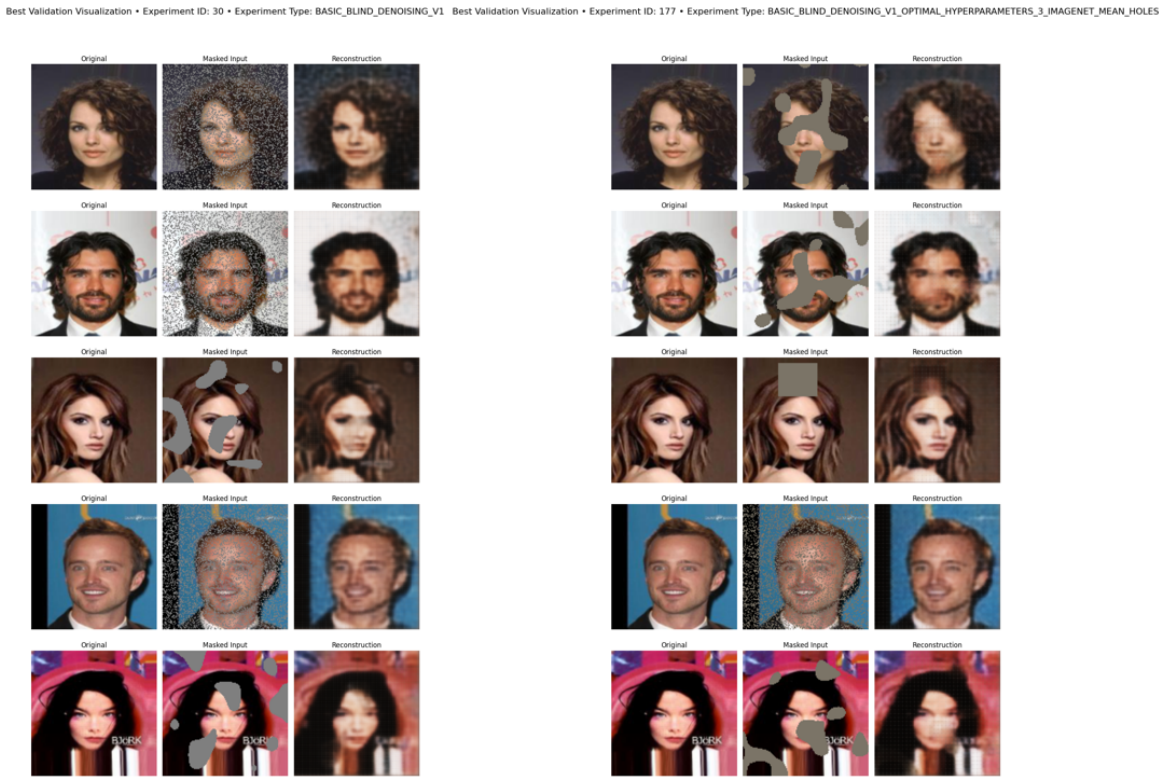


Figura 3: Comparación cualitativa entre inicialización de máscaras con valores RGB 127 (izquierda) y valor de la media de ImageNet (derecha).

9. **Barrido de tipos de máscaras en el modelo optimizado (Experimentos 181-210):** Teniendo definidos los valores de inicialización para los píxeles enmascarados, se realiza un barrido con cada tipo de máscara de forma independiente para valorar la efectividad del modelo ante cada tipo de enmascaramiento individual.

   Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la siguiente tabla. Como era de esperarse es más sencillo para el modelo procesar máscaras de tipo MCAR mientras que tiene más dificultad a la hora de procesar máscaras de tipo Pattern. Sorprendentemente, sin embargo, al entrenar únicamente con máscaras de tipo MCAR la desviación estándar aumenta, mientras que al entrenar únicamente con máscaras tipo Rectangular o

Pattern la desviación estándar disminuye, lo cual sugiere que para las máscaras tipo Mixture el aumento de la desviación estándar es principalmente provocado por las máscaras tipo MCAR.

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de ejecuciones	Desviación estándar de validación
Máscara Mix	0.18477706463550622 (id 177)	0.13734915451361582 (id 171)	15	0.005857868851381317
Máscara MCAR	0.1703739140253022 (id 182)	0.11989275791133061 (id 190)	10	0.005650239636291236
Máscara Rectangular	0.17768009019863307 (id 195)	0.13092156241719538 (id 196)	10	0.0031739583043695853
Máscara Pattern	0.1894753266055629 (id 202)	0.14279198818481886 (id 208)	10	0.002504748712512475

Tabla 9: Resultados cuantitativos del barrido de tipos de máscara.

10. **Cambio en la definición de la función de pérdida (Experimentos 211-240):** Habiendo llegado a los límites de esta arquitectura y con esta función de pérdida en particular, un ligero aumento de complejidad fue necesario. Basándose en el trabajo de Zhao et al. (2017) se crea una implementación de su función avanzada de pérdida, la cual combina la pérdida L1 aplicada no globalmente como se venía haciendo hasta ahora, sino mediante un núcleo gaussiano que evalúa el error región por región y al final obtiene un promedio del total, y una variante multiescala de SSIM conocida como MS-SSIM la cual evalúa el valor del error SSIM a múltiples resoluciones lo cual la vuelve menos sensible a sesgos uniformes en zonas planas o brillantes de la imagen. Con esto se define Mix Loss, cuya ecuación está definida por los autores como:

$$\mathcal{L}^{\text{Mix}} = \alpha \cdot \mathcal{L}^{\text{MS-SSIM}} + (1-\alpha) \cdot G^{M}_{\sigma_G} \cdot \mathcal{L}^{\ell_1},$$

A pesar de que no tiene sentido comparar los resultados cuantitativos de 2 funciones de pérdida diferentes se anexan en la tabla 9 los valores obtenidos, asimismo la figura 4 nos da una comparación cualitativa entre

la calidad de las reconstrucciones, la cual nos muestra que el cambio a Mix Loss hace una enorme diferencia incluso teniendo la misma arquitectura básica con los mismos hiperparámetros, este es uno de los puntos resaltados por Zhao et al. (2017) y abre pauta a que incluso mejores reconstrucciones sean posibles en experimentos futuros haciendo uso de funciones de error más complejas.

Tipo de Experimento	**Error mínimo de entrenamiento**	**Error mínimo de validación**	**Número de ejecuciones**	**Desviación estándar de validación**
Error Base	0.18477706463550622 (id 177)	0.13734915451361582 (id 171)	15	0.005857868851381317
Error Mix propuesto por Zhao et al. (2017)	0.0583454413579648 (id 219)	0.04124309382855128 (id 238)	30	0.000978595611227278

Tabla 10: Resultados cuantitativos de las funciones de pérdida.

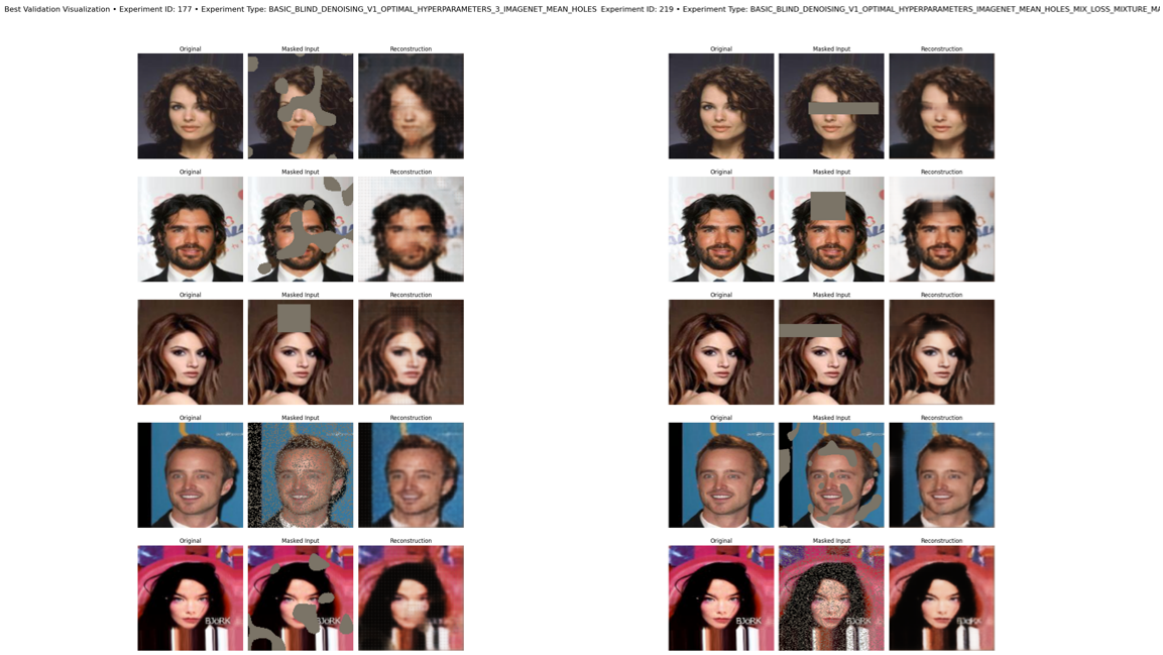


Figura 4: Comparación cualitativa entre función de pérdida original (izquierda) y función de pérdida propuesta por Zhao et al., 2017 (derecha)

11. **Barrido de interpolación y conexiones (Experimentos 241-270):** Una vez se definió que el nuevo modelo base tomaría como función de pérdida Mix Loss de Zhao et al. (2017). Se probó realizar cambios ligeros a la arquitectura del autocodificador de eliminación de ruido. El primer cambio probado fue utilizar interpolación bilineal (bilinear interpolation) en el proceso de decodificado en lugar de las capas convolucionales transpuestas. El segundo cambio consistió en probar el efecto que tendrían conexiones de salto simples (concatenación) y conexiones tipo U-Net entre el codificador y decodificador.

    Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la siguiente tabla.

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de ejecuciones	Desviación estándar de validación
Arquitectura Base	0.0583454413579648 (id 219)	0.04124309382855128 (id 238)	30	0.000978595611227278
Interpolación Bilineal	0.09696953992634637 (id 249)	0.0808279534849601 (id 246)	10	0.001435150515441518
Conexiones de salto concatenadas	0.041895791642227266 (id 255)	0.03201553827294937 (id 255)	10	0.0007123325893200051
Conexiones tipo U-Net	0.039472670417349295 (id 270)	0.02807455277070403 (id 263)	10	0.0004790398757491122

Tabla 11: Resultados cuantitativos del barrido de interpolación y conexiones.

Como puede apreciarse por los resultados de la tabla, el uso de interpolación bilineal en vez de convoluciones transpuestas resulta en un mayor error tanto de entrenamiento como de validación. Mientras tanto, el uso de conexiones de salto concatenadas mejora el desempeño cuantitativo, mientras que el uso de conexiones tipo U-Net presenta resultados incluso mejores. Sin embargo, hay un inconveniente en la implementación de estas conexiones. Es verdad que mejoran la calidad general de las reconstrucciones, ya que evitan la pérdida de información entre capas y eso resulta en una mayor fidelidad en las áreas no enmascaradas, el problema yace en que en este punto de la experimentación se está haciendo uso de inpainting ciego, es decir que el sistema no sabe explícitamente qué píxeles se encuentran enmascarados y cuáles no, esto resulta en un realce de las zonas enmascaradas y que estas sean considerablemente más visibles en la reconstrucción, como puede observarse en la siguiente figura.

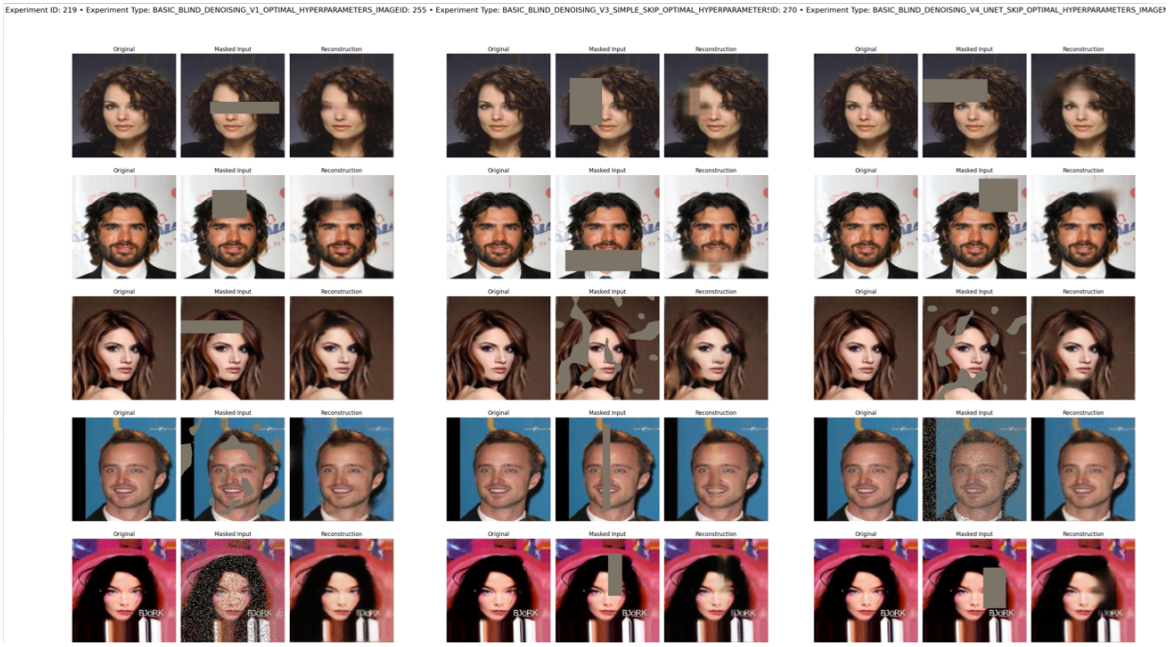


Figura 5: Comparación cualitativa entre modelo base (izquierda), conexiones de salto concatenadas (centro) y conexiones tipo U-Net (derecha)

12. **Barrido de tamaño de cuello de botella (Experimentos 271-300):** El tamaño del cuello de botella de un autocodificador de eliminación de ruido determina el tamaño mínimo al cual será comprimida la representación abstracta de una imagen dada. Un cuello de botella demasiado grande arriesga a preservar un porcentaje demasiado elevado de la información corrompida, mientras que un cuello de botella demasiado pequeño no le da suficiente información al decodificador a partir de la cual generar una reconstrucción adecuada.

    Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la siguiente tabla. En este caso el tamaño de cuello de botella de 384 filtros fue considerado como la alternativa más óptima debido a que su mejor resultado supera cuantitativamente tanto en error de entrenamiento como en error de validación al tamaño base de 256 filtros sin sacrificar estabilidad en el entrenamiento como es el caso del tamaño de 512 filtros.

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de ejecuciones	Desviación estándar de validación
Arquitectura Base (cuello de botella 256 filtros)	0.0583454413579648 (id 219)	0.04124309382855128 (id 238)	30	0.000978595611227278
Cuello de botella 128 filtros	0.06085371887369913 (id 280)	0.04289042763411999 (id 246)	10	0.00152355648613795
Cuello de botella 512 filtros	0.05606264094148511 (id 287)	0.03996562055097176 (id 290)	10	0.001352146455798899
Cuello de botella 384 filtros	0.05770505499094725 (id 270)	0.04110672451460209 (id 297)	10	0.0007737434024307274

Tabla 12: Resultados cuantitativos de la comparación entre distintos tamaños de cuello de botella.

En el caso de este barrido los resultados cualitativos indican que en contraste con el uso de conexiones de salto y conexiones tipo U-Net, aumentar el tamaño del cuello de botella no introduce artefactos en las zonas enmascaradas de forma tan sustancial, sin embargo, vuelve al sistema más sensible a las máscaras tipo MCAR, esto puede apreciarse en la siguiente figura.

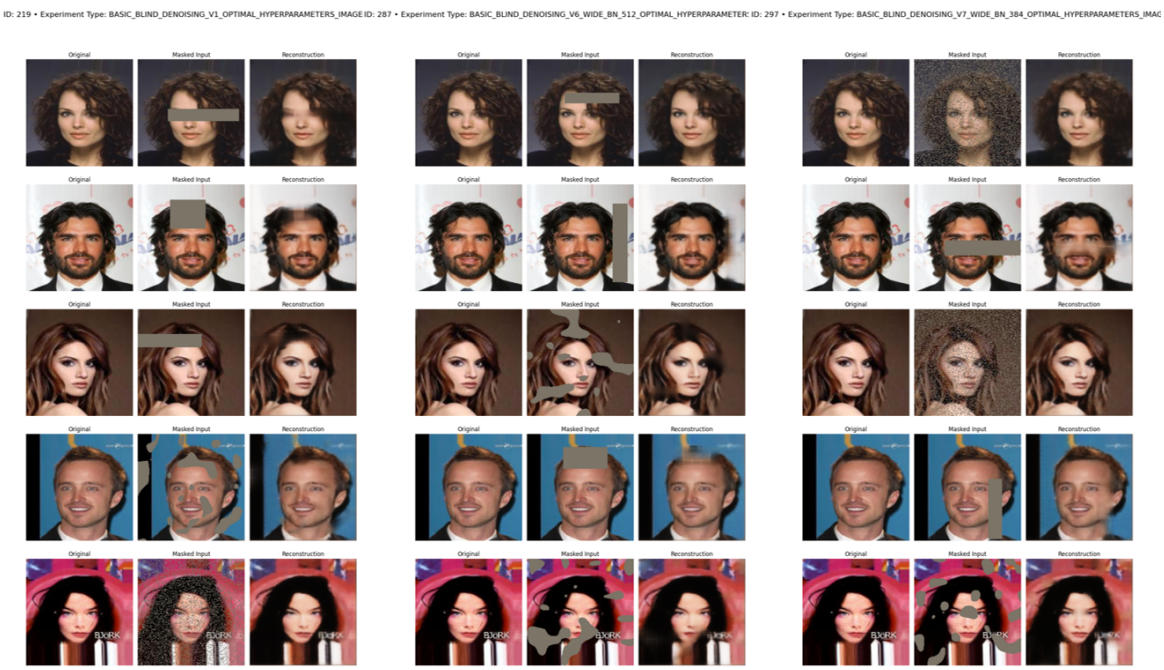


Figura 6: Comparación cualitativa entre modelo base (izquierda), cuello de botella de 512 filtros (centro) y 384 filtros (derecha)

13. **Barrido de profundidad de la red (Experimentos 301-320):** La profundidad de una red neuronal convolucional (CNN) influye no solamente en el número de parámetros y tiempo de inferencia, sino también en su capacidad de abstracción, una red más profunda le permitirá al modelo extraer características cada vez más abstractas y semánticamente significativas de la entrada, y en su sensibilidad al problema del desvanecimiento del gradiente (vanishing gradient problem), los gradientes de una red neuronal más profunda tienden al 0 debido a que siempre se multiplican por valores normalizados entre 0 y 1 o entre -1 y 1 debido al comportamiento de la normalización y las funciones de activación del sistema.

    Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la siguiente tabla:

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de ejecuciones	Desviación estándar de validación
Arquitectura Base (4 bloques convolucionales en codificador y 4 en decodificador)	0.0583454413579648 (id 219)	0.04124309382855128 (id 238)	30	0.000978595611227278
5 bloques convolucionales en codificador y 5 en decodificador	0.06536273803245628 (id 301)	0.04491328259404653 (id 310)	10	0.0009734646323509604
3 bloques convolucionales en codificador y 3 en decodificador	0.048193985865738406 (id 319)	0.038379911022881665 (id 319)	10	0.00035653751048690663

Tabla 13: Resultados cuantitativos del barrido de tamaño de profundidad de la red.

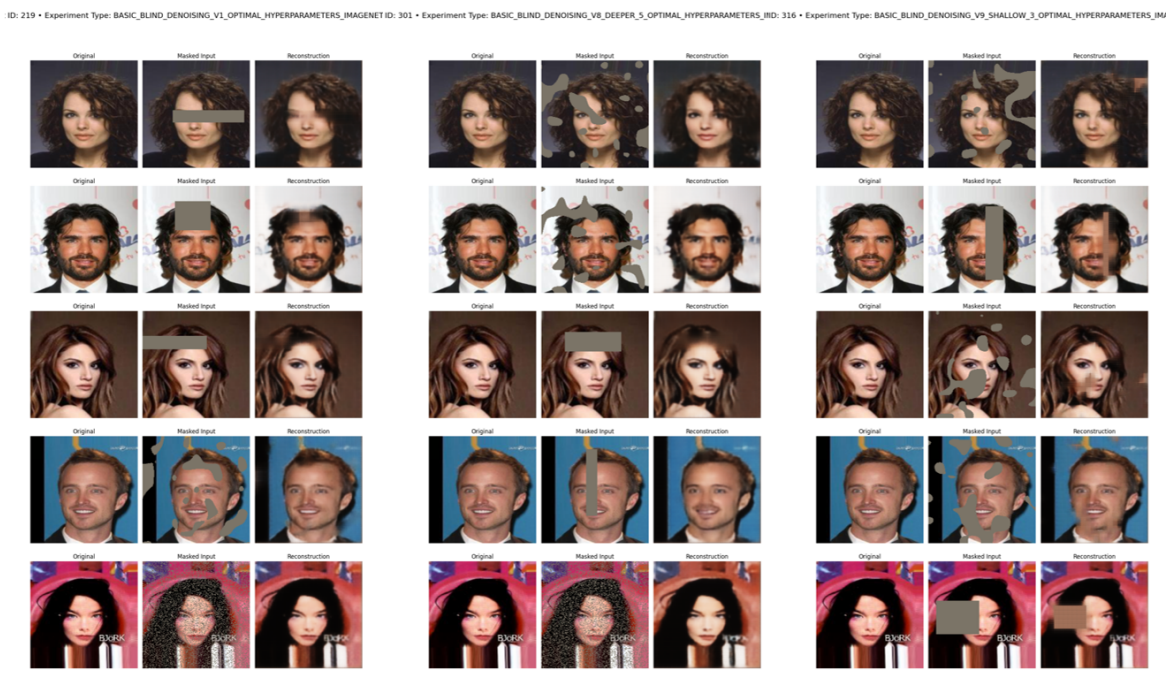


Figura 7: Comparación cualitativa entre modelo base (izquierda), modelo con 5 bloques convolucionales (centro) y 3 bloques convolucionales (derecha)

Como puede apreciarse por los resultados de la tabla, el uso de una red más profunda sin tomar las precauciones necesarias para evitar el problema del desvanecimiento del gradiente resulta en un aumento del error cuantitativo, así como un cambio más visible de color respecto a su contraparte de 4 bloques convolucionales, como puede apreciarse en la figura. Por otra parte la implementación de una red con solo 3 bloques convolucionales mejora en todos los aspectos los resultados cuantitativos respecto a la arquitectura base, sin embargo, esta mejoría es engañosa como puede observarse en los resultados cualitativos de la figura, ya que a pesar de que las áreas no enmascaradas son reconstruidas de mejor manera se generan artefactos considerablemente visibles en las áreas enmascaradas de la imagen, lo cual nos indica que el modelo es incapaz de aprender a ignorar la información corrompida con tan pocos bloques convolucionales. Esto nos da 2 revelaciones clave dada la información que hemos recopilado hasta este punto:

(a) Una red más profunda, puede ser una buena alternativa, si se combina con los mecanismos necesarios para evitar el problema del desvanecimiento del gradiente.

(b) Mejores resultados cuantitativos no necesariamente resultarán en mejores reconstrucciones para nuestro problema en particular, por lo cual es importante la combinación de criterios tanto cuantitativos como cualitativos para determinar el desempeño de nuestro sistema.

14. **Barrido de funciones de activación (Experimentos 321-350):** Las funciones de activación de las capas ocultas de toda red neuronal artificial, incluyendo las CNNs son determinantes para el comportamiento no lineal del sistema. Funciones de activación más complejas conllevan consigo un mayor tiempo de cómputo a cambio de una mayor expresividad de características y variedad de valores para diferentes entradas. Por otro lado, funciones de activación más sencillas tienden a reducir el tiempo de cómputo e inferencia para redes más profundas y complejas a coste de una menor expresividad de características.

    Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la siguiente tabla. En este caso la función de activación Parametric ReLU fue considerada como la alternativa más óptima debido a que fue capaz de superar al modelo base consistentemente tanto en error de entrenamiento como en error de validación, sin empeorar los resultados cualitativos (ver Figura).

Tipo de Experimento	Error mínimo de entrenamiento	Error mínimo de validación	Número de ejecuciones	Desviación estándar de validación
Arquitectura Base (tansig)	0.0583454413579648 (id 219)	0.04124309382855128 (id 238)	30	0.000978595611227278
ReLU	0.05373569476014038 (id 324)	0.03813128479016133 (id 328)	10	0.0005268598448543769
Leaky ReLU	0.053117619379300554 (id 340)	0.03776981211148012 (id 340)	10	0.0003720194377396071
Parametric ReLU (PReLU)	0.04875897600918821 (id 348)	0.03559704427010356 (id 341)	10	0.0003869161905344136

Tabla 14: Resultados cuantitativos del barrido de funciones de activación.

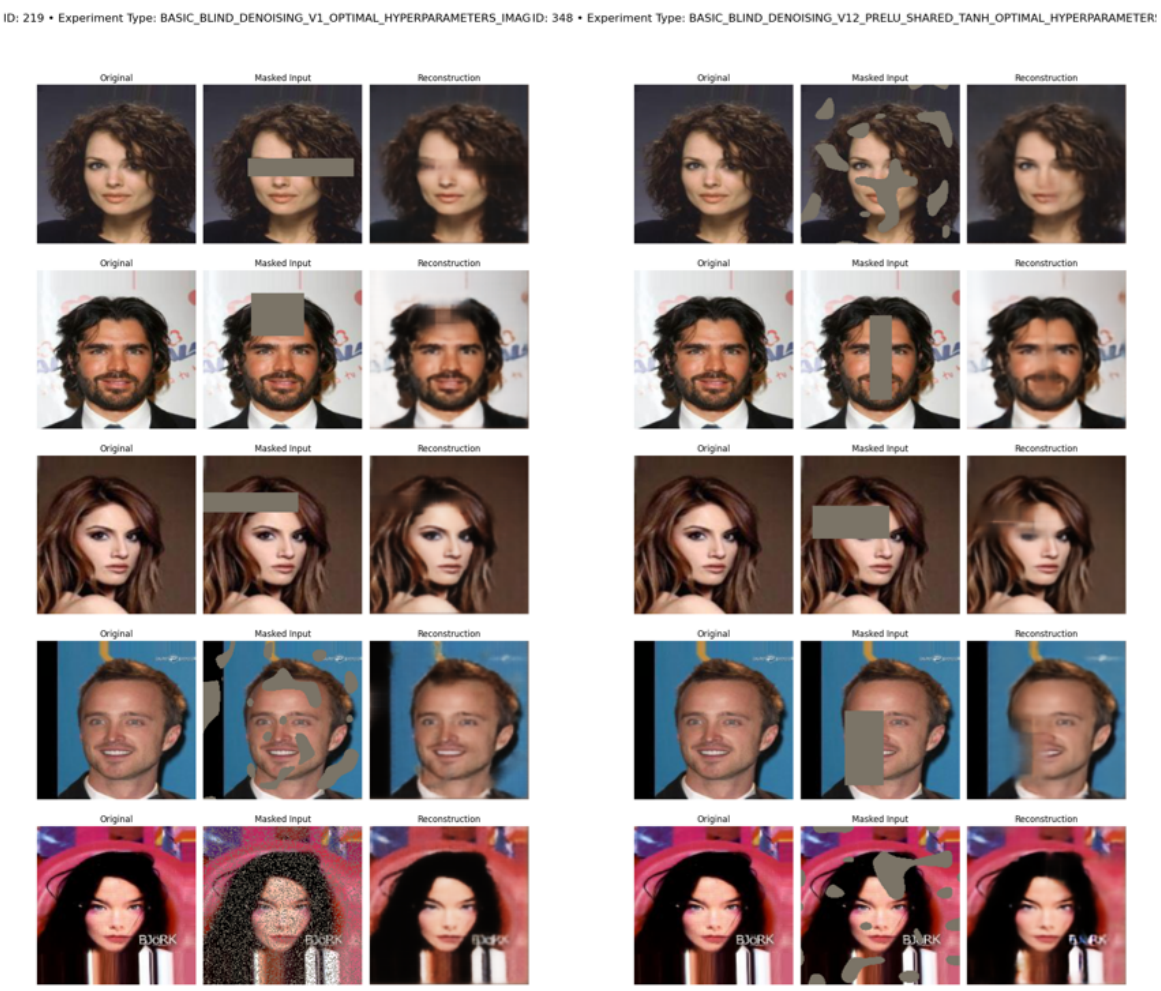


Figura 8: Comparación cualitativa entre modelo base tansig (izquierda) y modelo PReLU (derecha)

15. **Cambio a inpainting no ciego con mecanismos de atención básicos (Experimentos 351-370):** La implementación de mecanismos de atención al problema de inpainting consiste en indicarle al modelo de forma explícita qué regiones de la imagen de entrada se encuentran enmascaradas, al realizar esta distinción se dice que pasamos de un problema de inpainting ciego a un problema de inpainting no ciego. Existen todo tipo de mecanismos de atención para el inpainting no ciego y , de manera general, se considera que las reconstrucciones no ciegas tienden a ser

mucho mejores tanto cualitativamente como cuantitativamente que su contraparte ciega, ya que el modelo cuenta con mucha más información a partir de la cual realizar inferencias.

Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la siguiente tabla. En este caso se confirma que cambiar de un problema de inpainting ciego a no ciego, incluso siendo el mecanismo de atención el más básico posible, produce mejores resultados tanto cuantitativa como cualitativamente (ver Figura). Por otra parte, al sumar la mejora producida por los mecanismos de atención con la función de activación PReLU, se obtienen resultados incluso mejores. Es por esto último que a partir de este punto nuestro nuevo modelo base será el modelo no ciego por concatenación de máscaras con función de activación PReLU.

Tipo de Experimento	**Error mínimo de entrenamiento**	**Error mínimo de validación**	**Número de ejecuciones**	**Desviación estándar de validación**
Arquitectura Base (inpainting ciego)	0.0583454413579648 (id 219)	0.04124309382855128 (id 238)	30	0.000978595611227278
Inpainting no ciego (concatenación básica)	0.05545622665353073 (id 351)	0.0388317936553787 (id 353)	10	0.0005063912509525276
Inpainting no ciego (concatenación básica + PReLU)	0.0459208057771595 (id 366)	0.03304411442233966 (id 364)	10	0.0003136091689225673

Tabla 15: Resultados cuantitativos de la comparación entre modalidades de inpainting ciego y no ciego.

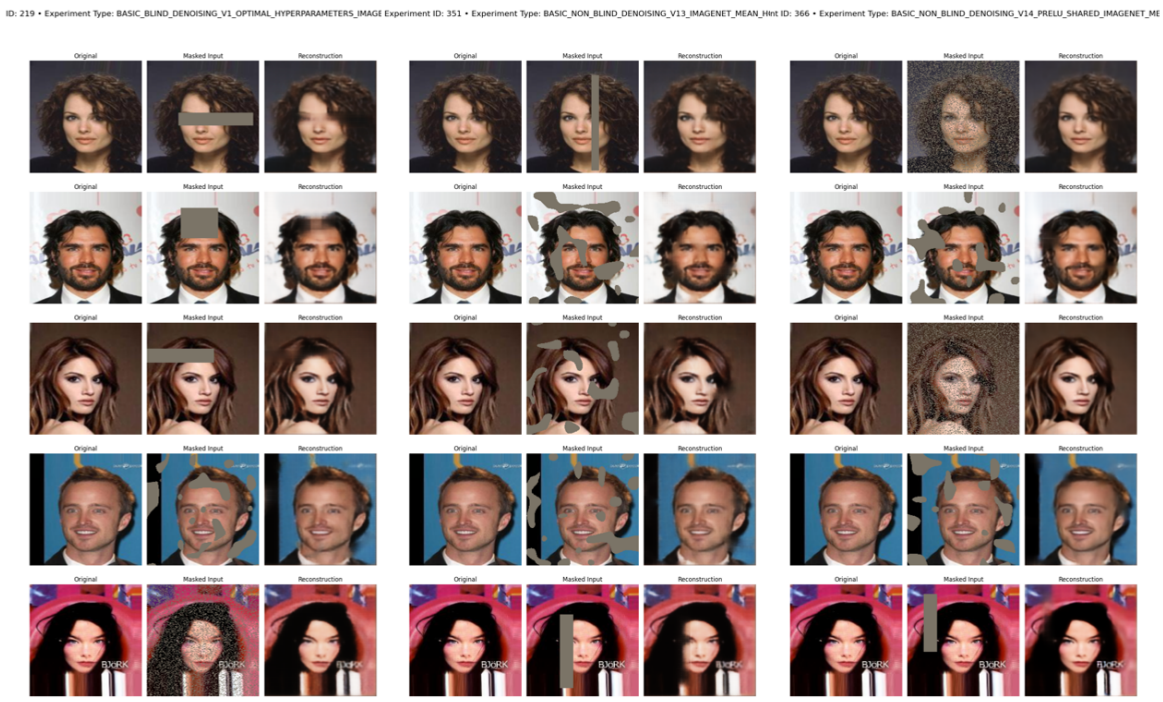


Figura 9: Comparación cualitativa entre modelo base (izquierda), modelo no ciego (centro) y modelo no ciego + PReLU (derecha)

16. **Barrido de tamaño de lote (Experimentos 371-400):** Como se ha discutido anteriormente, el proceso de aprendizaje de una red neuronal artificial se basa en el concepto de retropropagación y la regla de actualización de pesos. Comúnmente el entrenamiento del modelo requiere que este haga varios recorridos a través de todo el conjunto de datos de entrenamiento, a esto se le conoce como una época. Sin embargo, dentro de cada época el conjunto de datos puede ser dividido en un número de partes que puede ir desde 1 sola muestra por actualización de pesos (descenso de gradiente estocástico; stochastic gradient descent, SGD) hasta la totalidad del conjunto de datos; a este concepto se le conoce como tamaño de lote. Un tamaño de lote pequeño resultará en una mayor cantidad de actualizaciones de pesos por época, lo cual llevará más tiempo de cómputo, pero requerirá de menos memoria. Por otra parte, un tamaño de lote mayor resultará en una menor cantidad de ac-

tualizaciones de pesos por época, menos tiempo de cómputo, pero una demanda mayor de memoria.

Los resultados de este barrido (tomando el mejor experimento individual de cada conjunto) corresponden a la siguiente tabla. En este caso se puede apreciar que un menor tamaño de lote produce mejores resultados a coste de un mayor tiempo de entrenamiento. Sin embargo, la mejora de eficacia comienza a deteriorarse significativamente cuando pasamos de un tamaño de lote 32 a un tamaño de lote 16, el tiempo de cómputo aumenta sin proporcionar al modelo una mejoría significativa al error.

Tipo de Experimento	**Error mínimo de entrenamiento**	**Error mínimo de validación**	**Número de ejecuciones**	**Desviación estándar de validación**	**Tiempo promedio de entrenamiento (segundos)**
Arquitectura Base (tamaño de lote 128)	0.0459208057771595 (id 366)	0.03304411442233966 (id 364)	10	0.0003136091689225673	5239.522
Tamaño de lote 64	0.04452507520012511 (id 376)	0.03174850103891547 (id 371)	10	0.0004042282217794389	5539.689
Tamaño de lote 32	0.04325427053156081 (id 388)	0.03071173426849638 (id 388)	10	0.0003654853018879886	6141.66199999999
Tamaño de lote 16	0.04350951814253988 (id 393)	0.03075109376315621 (id 400)	10	0.0007262310915738061	6808.11999999999

Tabla 16: Resultados cuantitativos de barrido de tamaño de lote.

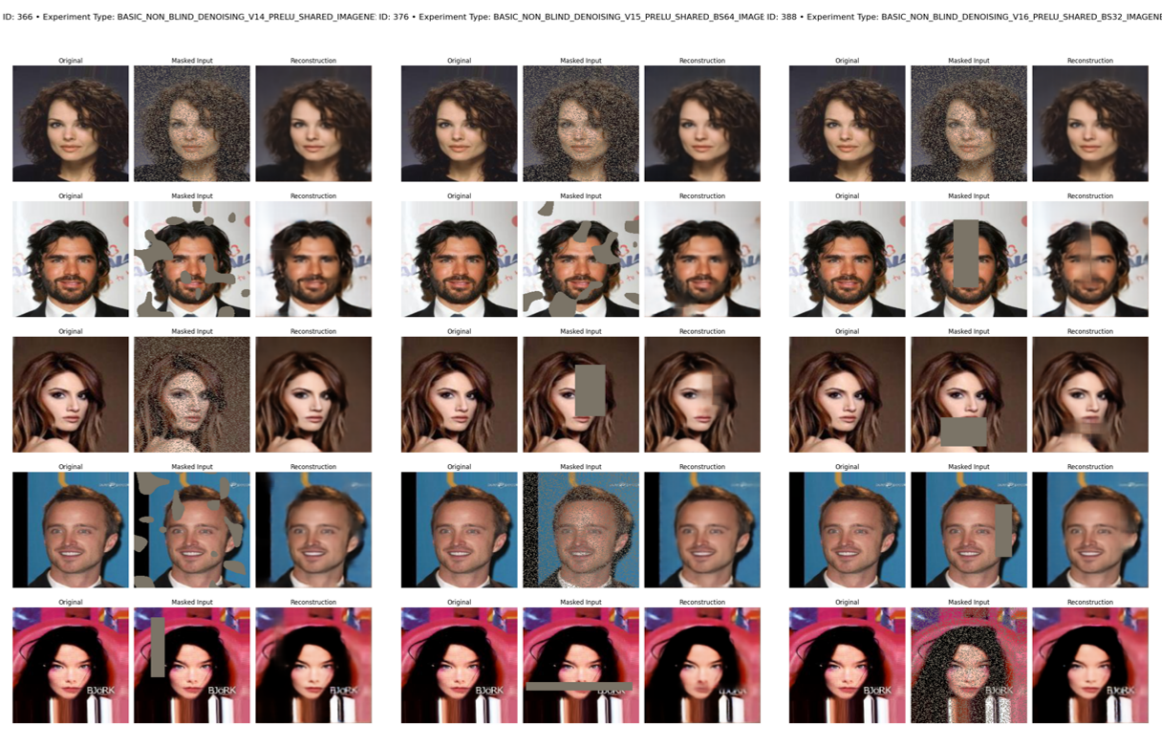


Figura 10: Comparación cualitativa entre modelo base con tamaño de lote 128 (izquierda), tamaño de lote 64 (centro) y tamaño de lote 32 (derecha)

3.3.3.1 Conclusiones de la primera iteración de entrenamiento

Gracias a la naturaleza repetitiva del entrenamiento de sistemas de redes neuronales y a la estructura modular con la que nuestro flujo de trabajo fue diseñado, la iteración a través de nuevos modelos y variantes se volvió mucho más ágil y eficiente a partir de este punto.

Tener la implementación de un sistema de máscaras dinámicas dentro del sistema dio la capacidad a este proyecto de ajustar la agresividad del entrenamiento para entrenar un modelo más robusto o más especializado de ser necesario.

Sin embargo, es importante analizar en qué medida los resultados preliminares mostrados hasta este punto del desarrollo son cualitativamente capaces de cumplir con el objetivo planteado para esta investigación: el poder

rellenar satisfactoriamente las secciones faltantes de un rostro humano a partir de los rasgos faciales principales de una persona.

Los modelos entrenados en los últimos barridos, aunque básicos, son capaces de rellenar secciones faltantes de cabello, piel y fondo de manera adecuada, sin embargo, todavía tienen problemas con estructuras faciales más complejas como lo es la reconstrucción de ojos, narices y bocas.

A pesar de que la reconstrucción de dichas secciones se considera formalmente fuera del alcance de la presente investigación, se consideró en este punto del desarrollo como un objetivo extra útil que permitiría a nuestro modelo obtener resultados más realistas en las áreas de enfoque principal en caso de mejorarse.

Por lo anteriormente mencionado se consideró necesario aumentar la complejidad del modelo para poder explorar estos objetivos adicionales. Esta conclusión preliminar llevó a la presente investigación a tomar 2 decisiones importantes que moldearían el desarrollo de la siguiente etapa:

1. **Agregar al flujo de trabajo la capacidad de comparar cuantitativamente los experimentos** El llevar a cabo una evaluación cuantitativa detallada de los resultados de cada modelo preliminar y no solo basarnos en los valores de la función de pérdida (los cuales no son comparables entre sí cuando parte del proceso iterativo incluye la capacidad de cambiarla) nos permitiría tener una decisión validada e informada en cuanto a la selección del modelo final, así como la posibilidad de comparar la calidad de nuestro trabajo con otros artículos similares.
2. **A partir de dichos datos explorar los límites de la arquitectura actual y la posibilidad de emplear otras arquitecturas** Los barridos realizados en esta primera etapa iterativa del desarrollo resultaron en mejorías significativas en la calidad de las reconstrucciones comparando con las primeras iteraciones.

Un modelo con la función de pérdida adaptada del trabajo de Zhao et al. (2017) combinada con una arquitectura U-NET, una generación de máscaras inspirada en el trabajo de Pathak et al. (2016) y normalización de píxeles basada en la media RGB de ImageNet se posicionó como una base fuerte para ajuste fino (fine-tuning) adicional, el explorar los límites de esta combinación para obtener un modelo de inpainting efectivo y con costes computacionales de entrenamiento y ejecución razonables se consideró como un siguiente objetivo importante en el desarrollo.

Al mismo tiempo se contempló la posibilidad de explorar otras arquitecturas presentadas en el estado del arte, entre las cuales los candidatos más destacables fueron:

(a) La arquitectura de convoluciones parciales presentada en el trabajo de Liu et al. (2018).

(b) La arquitectura de convoluciones de Fourier presentada en el trabajo de Suvorov et al. (2021).

(c) La arquitectura de Autocodificador Variacional con Condicionamiento Arbitrario presentada en el trabajo de Ivanov et al. (2019).

Habiéndose tomado estas decisiones el desarrollo continuó con la siguiente etapa.

3.4 Evaluación cuantitativa, ajuste fino y discusión de resultados

En esta sección se continuó con el ajuste fino de los modelos desarrollados durante la etapa 3.3.3, tomando como punto de partida la versión V26. A partir de esta arquitectura se entrenaron y evaluaron nuevas variantes para

mejorar sus resultados. A diferencia de la etapa anterior, el criterio cuantitativo para comparar las versiones pasó de la función de pérdida en validación a las métricas de calidad de imagen calculadas sobre las reconstrucciones obtenidas al realizar inferencia con los modelos entrenados, manteniendo la inspección visual de las reconstrucciones como un criterio complementario de selección.

Al igual que en la etapa 3.3.3 se llevó un registro del desempeño de estas variantes del modelo y los registros fueron almacenados en una base de datos extensiva para su consulta abierta.

La comparación de resultados de inpainting entre diferentes modelos puede sonar en primera instancia como un proceso simple y directo, sin embargo, como se demuestra en la sección 3.3.3, en un problema visual como lo es el inpainting de imágenes, mejores resultados cuantitativos pueden producir resultados cualitativos no deseados.

Adicionalmente el problema del inpainting puede de cierta forma ser considerado no determinístico, ya que existen numerosas combinaciones de valores viables para rellenar una sección faltante en una imagen de forma que se vea “natural” para un ser humano. Suvorov et al. (2021) expresa que “el problema del inpainting es inherentemente ambiguo, existen muchos rellenados plausibles para la misma región faltante, especialmente cuando los agujeros se vuelven grandes”. Liu et al. (2018) secunda esta noción “No existe una métrica numérica correcta para evaluar los resultados del inpainting de imágenes debido a la existencia de muchas posibles soluciones”.

Habiendo expresado esto, en la literatura sí existen algunos parámetros cuantitativos estándar para la evaluación de estos modelos, solo hay que recordar que por sí mismos estos no garantizan que el modelo sea bueno para el juez final, el ser humano. Burlin et al. (2017) reporta el error L2. Xie et al. (2012) reporta el PSNR. Zhao et al. (2017) reporta el error L1, L2, PSNR, SSIM, MS-SSIM, IW-SSIM, FSIM y GMSD. Liu et al. (2018) reporta el error L1, PSNR, SSIM e IScore. Ivanov et al. (2019) reporta usar el PSNR.

Youngjoo Jo et al. (2019) reporta una pérdida adversarial tipo GAN. Zhou et al. (2020) reporta el error L1, PSNR, SSIM y LPIPS. Lee et al. (2020) reporta el FID. Suvorov et al. (2021) reporta el FID y el LPIPS. Chandni Agarwal et al. (2023) reporta el error L1, FID, PSNR y SSIM.

De los ejemplos anteriores reportados en la literatura podemos establecer una base a partir de las métricas más comunes: el error L1, el error L2, el PSNR, el SSIM, la FID y el LPIPS.

Para la evaluación final del mejor modelo se incorporó LPIPS al flujo de trabajo de evaluación utilizando la variante basada en AlexNet. Debido al costo computacional adicional de esta métrica, su cálculo se realizó únicamente para la mejor arquitectura sobre el 25% del conjunto de validación de CelebA, equivalente a 4,967 imágenes. Al igual que FID, LPIPS se calculó tanto para la imagen completa como para la región reconstruida.

3.4.1 Desafíos principales del análisis cuantitativo de resultados

Habiendo establecido una base cuantitativa estandarizada a partir de la literatura, es importante destacar que el comparar modelos desarrollados en condiciones y arquitecturas heterogéneas conlleva consigo una serie de retos que deben considerarse a la hora de implementar la metodología de análisis de resultados. Los retos principales de esta tarea consisten en:

1. **Heterogeneidad arquitectónica:**
    (a) Las variantes de inpainting ciego y no ciego cuentan con diferentes dimensiones en cuanto a sus canales de entrada.
    (b) Diferentes arquitecturas cuentan con diferentes profundidades lo cual conlleva asimismo la necesidad de aumentar la dimensión mínima de las imágenes de entrada para las arquitecturas más profundas.

(c) Distintos bloques/funciones de activación conllevan la necesidad de poder definir el rango de normalización a la entrada, ya sea [0,1] o [-1,-1].

2. **Variabilidad de pérdidas:** Diferentes modelos a lo largo del desarrollo del presente proyecto fueron definidos con distintas funciones de pérdida, por lo cual una comparación directa entre los valores de pérdida en el set de validación no aporta información relevante para comparar la efectividad de un modelo frente a otro.

3. **Diversidad de máscaras:** Diferentes modelos presentarán un desempeño diferente frente a distintos tipos de máscaras, la metodología debe ser capaz de ajustar el patrón de evaluación para obtener una mejor intuición acerca de las fortalezas y debilidades de los modelos.

4. **Reproducibilidad:** Las evaluaciones deben ser consistentes e independientes de los programas/entornos de entrenamiento. Sin una buena capacidad de reproducibilidad los modelos resultarán obsoletos fuera del entorno de pruebas.

3.4.2 Objetivos específicos del sistema de evaluación

Tomando en cuenta los retos anteriormente mencionados se definieron objetivos específicos que el sistema de evaluación debía ser capaz de cumplir:

1. Proveer métricas comparables entre diferentes tipos de experimentos.

2. Desacoplar el flujo de evaluación de las configuraciones de entrenamiento para así poder poner a prueba un mismo modelo bajo diferentes escenarios.

3. Mantener registros completos de hiperparámetros y del contexto de ejecución. De esta forma se puede llevar registro de los escenarios

bajo los cuales los modelos fueron evaluados, incluso si las métricas son las mismas.

4. Generar visualizaciones reproducibles para realizar un análisis cualitativo. Ya que el inpainting es un problema no determinístico y altamente subjetivo a la percepción humana, no solo es importante poder evaluar cuantitativamente las imágenes obtenidas, también resulta necesaria una inspección cualitativa para detectar artefactos, desenfoques y otros fenómenos visuales que no necesariamente se traducen notablemente a las métricas cuantitativas.

5. Facilidad de consultas mediante persistencia en base de datos. Al lidiar con grandes volúmenes de datos y conforme la cantidad de registros fuese creciendo se volvería más desafiante comparar la información recopilada a menos que se mantuviera una estructura estandarizada con campos indexados cuyo acceso y consulta resultaran convenientes para el investigador, es por esto que se planteó definir funciones programáticas para las consultas más comunes e importantes.

Para cumplir con estos objetivos, se diseñó y desarrolló el siguiente sistema de evaluación semiautomática:

3.4.3 Arquitectura del sistema de evaluación

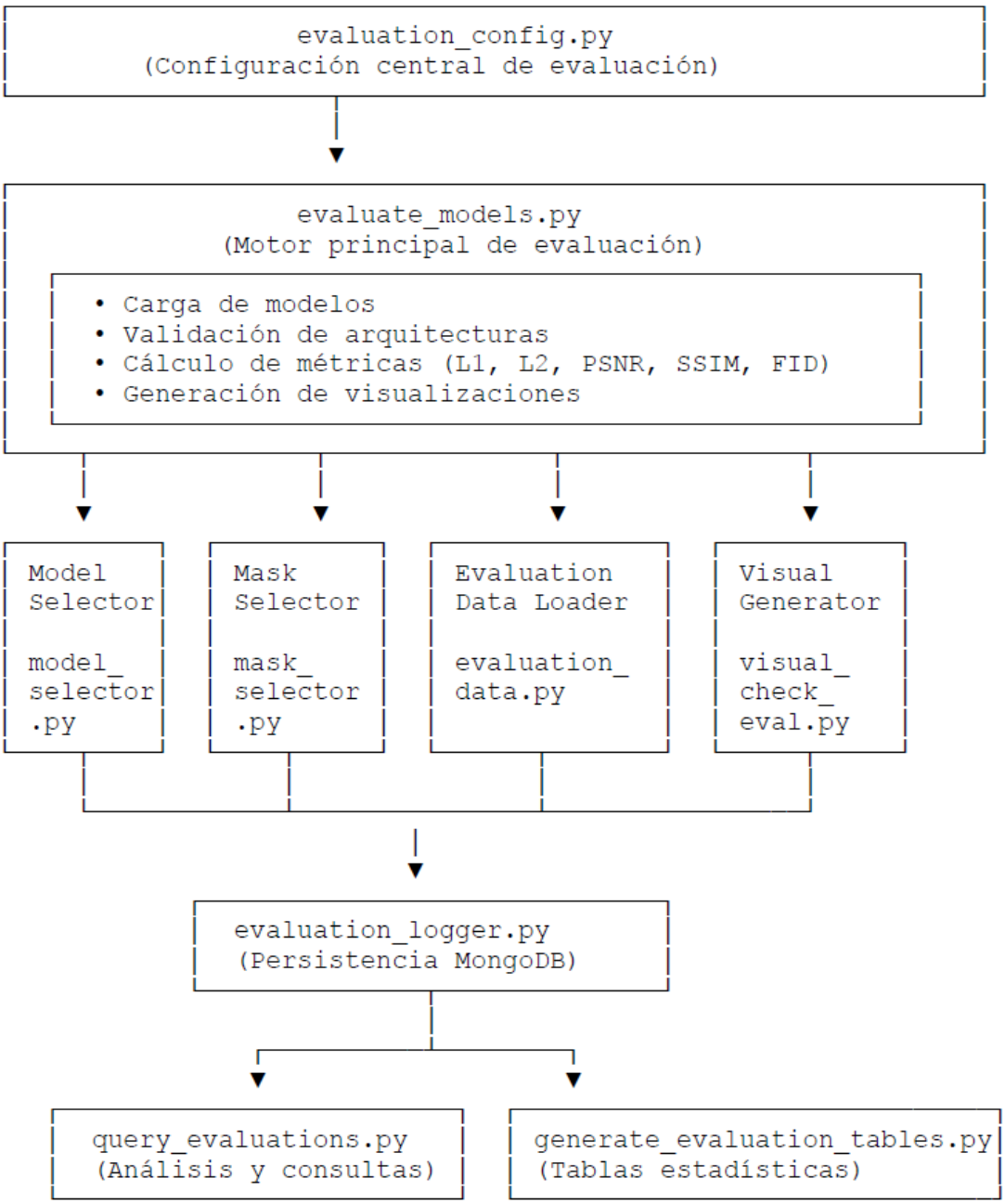


Figura 11: Componentes principales de la arquitectura del sistema de evaluación

La figura muestra visualmente los componentes principales del sistema de evaluación cuantitativo, así como su flujo de datos, los detalles de su funcionamiento se detallan a continuación:

1. **Definición de configuración (evaluation_config.py):** Se especifican uno o varios modelos a evaluar. Esta configuración puede ser extraída desde los registros existentes en la base de datos de MongoDB (de forma automática), o puede ser definida explícitamente en un diccionario local (de forma manual).

2. **Orquestación (evaluate_models.py):** Se lee la configuración y se itera sobre cada entrada del diccionario; se fija el dispositivo (ya sea CPU o GPU), se instancian el cargador de datos, el selector de modelos y el generador de máscaras.

3. **Selección de modelo (model_selector.py):** A partir del nombre de la arquitectura se instancia la clase (arquitectura) y se cargan los pesos desde un archivo. pth. Se valida la coherencia de canales a la entrada (3 para inpainting ciego, 4 para inpainting no ciego). Y se valida que la arquitectura concuerde con la cantidad de parámetros del archivo .pth.

4. **Carga de los datos de validación (evaluation_data.py):** Se construye un DataLoader, una función de apoyo que divide el conjunto de datos de validación en lotes de un tamaño determinado, se ajusta la resolución de las imágenes, se convierten las imágenes a tensores interpretables por PyTorch y se normalizan sus valores a 2 rangos configurables, ya sea [-1,1] o [0,1] dependiendo de las necesidades de la arquitectura y sus respectivas funciones de transferencia.

5. **Enmascaramiento de imágenes (mask_selector.py + mask_config.py):** Se crea un generador de máscaras en el cual se puede elegir entre las siguientes opciones: mixture, mcar, rectangle, pattern. Esto

permite analizar el comportamiento y desempeño de los modelos frente a diferentes tipos de máscaras.

6. **Inferencia y métricas (evaluate_models.py):** Las imágenes se cargan desde el DataLoader al modelo y se calcula el promedio de las métricas de calidad de imagen: L1, L2, PSNR, SSIM, FID y, cuando se habilita, LPIPS sobre el conjunto de validación. En el caso de LPIPS el sistema permite definir la red base a utilizar, como AlexNet, y la fracción del conjunto de validación sobre la cual se calculará la métrica.

7. **Visualización (visual_check_eval.py):** Se generan imágenes comparativas divididas en 3 secciones: Original | Masked | Reconstructed, dichas imágenes son guardadas tanto localmente como en Google Drive.

8. **Persistencia de datos (evaluation_logger.py):** Se almacenan en la base de datos de MongoDB las métricas previamente calculadas, la configuración y los metadatos bajo los cuales la evaluación fue ejecutada.

9. **Análisis y consulta de resultados (query_evaluations.py):** A través de un conjunto de funciones el sistema permite listar evaluaciones específicas por título, obtener la mejor evaluación para una métrica dada, comparar por tipo de experimento y exportar los registros a formato CSV.

10. **Tablas Estadísticas (generate_evaluation_tables.py):** Haciendo uso del sistema de consulta en MongoDB descrito anteriormente se obtienen los registros de todas las evaluaciones y se agrupan por tipo de experimento, se cuenta la cantidad de registros realizados para cada tipo y se extraen sus métricas de evaluación para calcular la media y la desviación estándar de cada una, se obtienen como resultado tablas estadísticas creadas en Matplotlib lo cual permite cuantificar la variabilidad de resultados para una arquitectura debido a la aleatoriedad en el proceso de generación de máscaras.

Variables necesarias para configurar la evaluación de un modelo

1. **experiment_type: str** — Título de la evaluación a ejecutar.
2. **model_path: str** — Ruta al archivo .pth con los pesos del modelo.
3. **architecture: str** — Identificador único de la arquitectura del modelo.
4. **input_channels: int** — Número de canales de entrada 3 (inpainting ciego) o 4 (inpainting no ciego).
5. **image_size: int** — Resolución de las imágenes a la entrada del modelo.
6. **use_imagenet_mean: bool** — Determina si se utilizará el valor de la media RGB ImageNet o un valor 0 para rellenar los píxeles enmascarados
7. **normalization: ”0_1” | ”-1_1”** — Rango de normalización a la entrada del modelo.
8. **mask_type: ”mixture” | ”mcar” | ”rectangle” | ”pattern”** — Determina el tipo de enmascaramiento que se utilizará para la evaluación.
9. **lpips_enabled: bool** — Determina si se calculará la métrica LPIPS durante la evaluación.
10. **lpips_net: str** — Define la red base utilizada para el cálculo de LPIPS, por ejemplo AlexNet.
11. **lpips_dataset_fraction: float** — Indica la fracción del conjunto de validación utilizada para calcular LPIPS.
12. **lpips_random_seed: int** — Fija la semilla utilizada para seleccionar de forma reproducible el subconjunto de validación usado por LPIPS.

Modos de Operación:

1. **Modo Automático** (USE_AUTOMATIC_CONFIG = True):

    (a) Consulta MongoDB para extraer las variables de evaluación almacenadas durante el entrenamiento

    (b) Garantiza consistencia y estabilidad durante la evaluación ya que recrea las mismas condiciones bajo las cuales fue entrenado el modelo.

    (c) Requiere especificar el título del experimento a evaluar.

2. **Modo Manual** (USE_AUTOMATIC_CONFIG = False):

    (a) Utiliza configuraciones detalladas por el usuario en un diccionario.

    (b) Su principal uso yace en la evaluación de modelos antiguos (aquellos entrenados antes de centralizar las variables de evaluación en un solo campo).

    (c) Permite además evaluar variaciones de modelos conocidos cuando se desea observar su comportamiento en condiciones diferentes a las de su entrenamiento

Ejemplo de Registro de Evaluación en MongoDB


```
{
    "_id": ObjectId("6a31aace89d634443d5f310f"),
    "model_name": "best_model_NONBLIND_V28_UNET_NON_SHARED_PRELU6B_MIX_LOSS",
    "model_path": "./checkpoints/best_model_NONBLIND_V28_UNET_PRELU6B_MIX_LOSS.pth",
    "experiment_type": "V28_FINAL",
    "date_evaluated": ISODate("2026-06-16T13:58:06.484Z"),
    "metrics": {
        "l1_loss": 0.022263988293107485,
        "l2_loss": 0.0026850895200677565,
        "psnr": 31.781496780506078,
        "ssim": 0.9521763048310211,
        "fid_full": 1.3598726915012094,
        "fid_masked": 12.545899359745079,
        "lpips_full": 0.04398845956014545,
        "lpips_inpainted": 0.03138795897493644
```

```
    },
    "model_configuration": {
        "architecture": "ConvAutoencoderUNetSkipNonBlind384PReLUDeeper6BlocksNonShared",
        "input_channels": 4,
        "image_size": 256,
        "is_blind": false,
        "use_imagenet_mean": true,
        "normalization": "-1_1",
        "mask_type": "pattern",
        "mask_seed": 1375514612,
        "lpips_enabled": true,
        "lpips_net": "alex",
        "lpips_dataset_fraction": 0.25,
        "lpips_random_seed": 42
    }
}
```


3.4.4 Complementariedad de Métricas

La combinación de estas métricas nos proporciona una evaluación multidimensional:

Métrica	Sensibilidad	Interpretación	Uso Principal	Rango
L1	Errores promedio	Magnitud de error por píxel	Optimización de detalles	$[0, \infty)$
L2	Errores grandes	Severidad de los artefactos	Optimización de valores atípicos y artefactos	$[0, \infty)$
PSNR	Relación Señal/Ruido	Calidad global (dB)	Evaluación comparativa (benchmark) con la literatura	10–40 dB (10 dB = pobre, 40 dB = excelente)
SSIM	Estructura perceptual	Similitud Visual	Calidad perceptual, detección de blur	$[-1, 1]$
FID (imagen completa)	Distribución de características global	Similitud semántica global	Evaluación comparativa con la literatura	$[0, \infty)$
FID (regiones enmascaradas)	Distribución de características de las regiones enmascaradas	Similitud semántica de las regiones relevantes	Medir específicamente la calidad de inpainting	$[0, \infty)$
LPIPS (imagen completa)	Diferencias perceptuales en mapas de características	Similitud perceptual global	Evaluación perceptual de la reconstrucción completa	$[0, \infty)$
LPIPS (región reconstruida)	Diferencias perceptuales en la región reconstruida	Similitud perceptual de las regiones relevantes	Medir la apariencia de la zona rellenada	$[0, \infty)$

Tabla 17: Complementariedad de las métricas de calidad de imagen calculadas.

Ejemplo de Análisis Conjunto:

Modelo A: L1=0.010, PSNR=32.5 dB, SSIM=0.85, FID=45.2, LPIPS=0.070 ← Bueno en reconstrucción a nivel de píxel

Modelo B: L1=0.012, PSNR=31.2 dB, SSIM=0.92, FID=28.7, LPIPS=0.044 ← Mayor error en reconstrucción a nivel de píxel

Interpretación: Modelo B preserva mejor estructura perceptual (SSIM, FID, LPIPS) a pesar de presentar un mayor error numérico a nivel de píxel. FID bajo indica que las reconstrucciones son semánticamente más realistas, mientras que LPIPS bajo indica mayor cercanía perceptual entre la imagen original y la reconstruida.

3.4.5 Resultados de métricas de calidad de imagen y ajuste fino

Cuarto Mejor modelo (UNET Versión 26)

Para poner a prueba el sistema de evaluación cuantitativa y experimentar en los límites de la arquitectura UNET con convoluciones simples mediante ajuste fino, se decidió tomar como base el modelo con los mejores resultados cualitativos y de función de pérdida de la sección 3.3.3. Se definen a continuación los detalles de esta arquitectura (UNET versión 26):

- Codificador: Conformado por 6 bloques convolucionales cada uno con los siguientes submódulos:
  - Capa Convolucional (Conv2d):
    - Tamaño de núcleo (3x3)
    - Paso 2
    - Relleno 1

        - Codificador 1: 4 IN, 32 OUT
        - Codificador 2: 32 IN 64 OUT
        - Codificador 3: 64 IN 96 OUT
        - Codificador 4: 96 IN 128 OUT
        - Codificador 5: 128 IN 256 OUT
        - Codificador 6: 256 IN 384 OUT
    - Normalización por Lote (BatchNorm2d)
    - Función de activación PReLU (parámetro compartido entre todas las capas)
    - Dropout:
        - Probabilidad 0.1

- Decodificador: Conformado por 6 bloques deconvolucionales cada uno (con la excepción del decodificador 6) con los siguientes sub-módulos:

    - Capa Convolucional Transpuesta (ConvTranspose2d)
        - Tamaño de núcleo (3x3)
        - Paso 2
        - Relleno 1
        - Decodificador 1: 384 IN, 256 OUT
            - Conexión de salto a la salida con Codificador 5
        - Decodificador 2: 256 IN 128 OUT
            - Conexión de salto a la salida con Codificador 4
        - Decodificador 3: 128 IN 96 OUT
            - Conexión de salto a la salida con Codificador 3
        - Decodificador 4: 96 IN 64 OUT
            - Conexión de salto a la salida con Codificador 2
        - Decodificador 5: 64 IN 32 OUT

            - Conexión de salto a la salida con Codificador 1
        - Decodificador 6 (capa de salida): 32 IN 3 OUT
            - Sin conexión de salto
            - Función de activación Tanh
    - Normalización por Lote (BatchNorm2d)
    - Bloque de mezcla (mediante Conv2d) (núcleo 3×3, paso 1, relleno 1) con el mapa de características de su conexión de salto correspondiente.
    - Función de activación PReLU (parámetro compartido entre todas las capas)
    - Dropout:
        - Probabilidad 0.1

- Preprocesamiento de imágenes:
    - Ajuste de tamaño (192x192)
    - Conversión a tensores pytorch
    - Normalización [-1 1]

- Hiperparámetros:
    - Número de épocas: 50
    - Tamaño de lote: 32
    - Tasa de aprendizaje: 0.0005
    - Probabilidad de Dropout: 0.1
    - Optimizador: AdaBelief
        - β_1 0.9 β_2 0.95
        - decaimiento de pesos 0.00001
    - Inicialización de pesos: Xavier Normal
    - Enmascaramiento: Pattern

  - Escalado dinámico de gradientes: Activado
  - Función de pérdida Mix Loss
  - Planificador: ReduceLROnPlateau
    - Mode: min
    - Patience: 1
    - Factor: 0.5
    - Threshold: 0
    - Threshold_mode: “abs”
- Resultados:
  - Tiempo promedio de entrenamiento: 28744.5 segundos
  - Error mínimo de entrenamiento: 0.02589579886006763 (id 447)
  - Error mínimo de validación: 0.017301843689091732 (id 447)
- Número de Parámetros:
  - Total de parámetros: 4,467,748
  - Parámetros PReLU: 1

Resultados de métricas de calidad de imagen (UNET Versión 26)

Ejecución	Error L1	Error L2	PSNR (dB)	SSIM	FID (imagen completa)	FID (más-cara)
Corrida 1	0.030060	0.004295	29.7422	0.9349	2.6346	18.5570
Corrida 2	0.030083	0.004298	29.7387	0.9348	2.6340	18.8068
Corrida 3	0.030086	0.004306	29.7303	0.9348	2.6509	18.9453
Corrida 4	0.030100	0.004318	29.7234	0.9348	2.6430	18.7971
Corrida 5	0.030035	0.004269	29.7686	0.9350	2.6398	18.8102
Corrida 6	0.030091	0.004300	29.7394	0.9348	2.6410	18.6995
Corrida 7	0.030080	0.004299	29.7384	0.9348	2.6414	18.7538
Corrida 8	0.030077	0.004296	29.7420	0.9348	2.6296	18.9481
Corrida 9	0.030083	0.004304	29.7367	0.9348	2.6256	18.7527
Corrida 10	0.030050	0.004299	29.7400	0.9350	2.6512	18.7893
Media ± Std	**0.030075 ± 0.000019**	**0.004298 ± 0.000012**	**29.7400 ± 0.0110**	**0.9349 ± 0.0001**	**2.6391 ± 0.0079**	**18.7860 ± 0.1072**

Tabla 18: Estadísticas de evaluación del modelo best_model_NONBLIND_V26_UNET_PRELU6B_MIX_LOSS (10 ejecuciones).

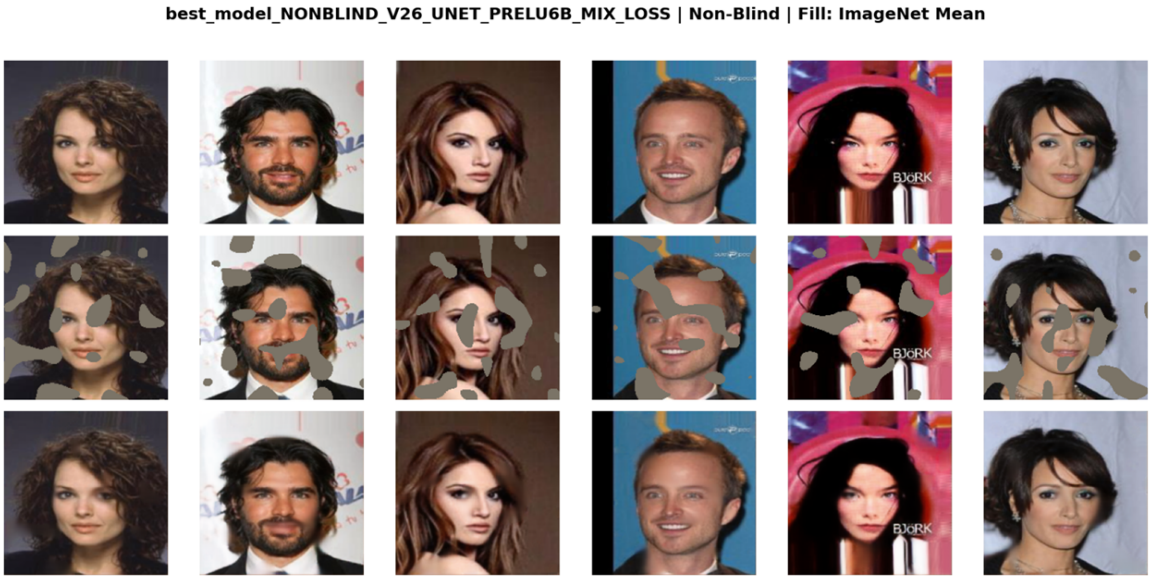


Figura 12: Resultados cualitativos de inpainting (UNET Versión 26)

Tercer Mejor modelo (UNET Versión 27)

Al notar durante la sección 3.3.3 que una mayor profundidad de capas resultaba en mejores resultados tanto cuantitativos como cualitativos se estableció un modelo UNET de 7 codificadores y 7 decodificadores, manteniendo el tamaño de cuello de botella en 384 filtros, el único otro cambio relevante de esta arquitectura frente a la anterior es el hecho de que se aumentó la resolución de las imágenes de entrada a 256x256 ya que de otra forma los kernels de convolución sobrecomprimían las imágenes y el modelo colapsaba.

- Resultados:
  - Tiempo promedio de entrenamiento: 34623.1 segundos
  - Error mínimo de entrenamiento: 0.022789466011765524 (id 455)
  - Error mínimo de validación: 0.014206586459537795 (id 455)

- Número de Parámetros:
  - Total de parámetros: 5,869,924
  - Parámetros PReLU: 1

Resultados de métricas de calidad de imagen (UNET Versión 27)

Ejecución	Error L1	Error L2	PSNR (dB)	SSIM	FID (imagen completa)	FID (máscara)
Corrida 1	0.024953	0.003051	31.2238	0.9450	1.6595	13.5112
Corrida 2	0.024945	0.003037	31.2444	0.9450	1.6588	13.3727
Corrida 3	0.024902	0.003031	31.2516	0.9452	1.6611	13.4240
Corrida 4	0.024947	0.003045	31.2306	0.9450	1.6557	13.3777
Corrida 5	0.024948	0.003045	31.2326	0.9450	1.6776	13.3650
Corrida 6	0.024951	0.003047	31.2316	0.9450	1.6749	13.3804
Corrida 7	0.024971	0.003056	31.2168	0.9449	1.6500	13.5109
Corrida 8	0.024930	0.003038	31.2444	0.9451	1.6520	13.5099
Corrida 9	0.025001	0.003061	31.2099	0.9448	1.6764	13.5043
Corrida 10	0.024958	0.003051	31.2215	0.9450	1.6597	13.5321
Media ± Std	**0.024951 ± 0.000024**	**0.003046 ± 0.000009**	**31.2307 ± 0.0126**	**0.9450 ± 0.0001**	**1.6626 ± 0.0096**	**13.4488 ± 0.0669**

Tabla 19: Estadísticas de evaluación del modelo best_model_NONBLIND_V27_UNET_PRELU7B_MIX_LOSS (10 ejecuciones).

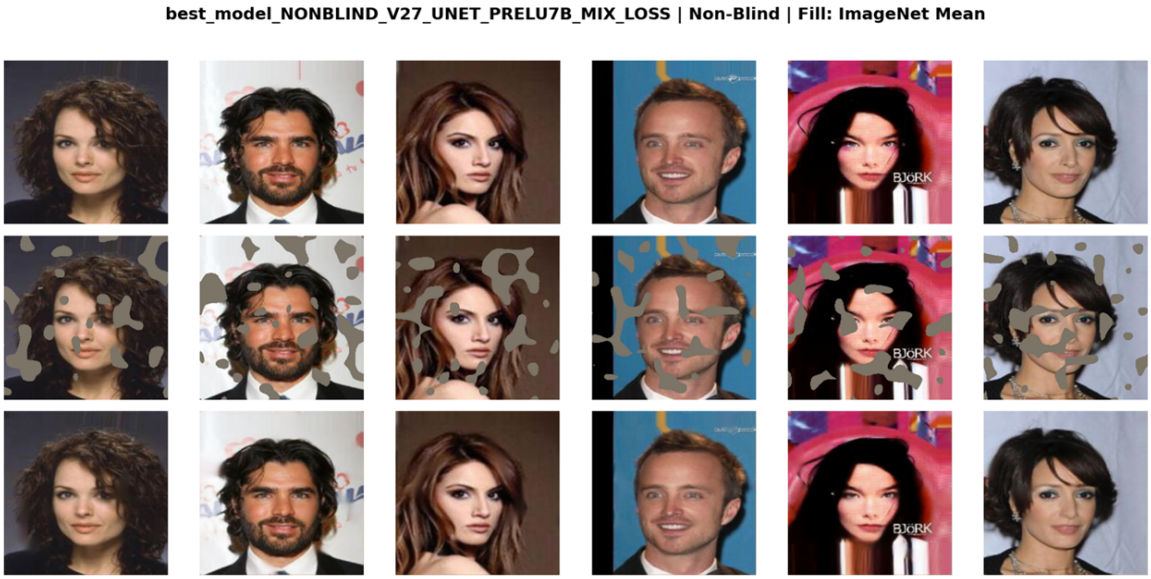


Figura 13: Resultados cualitativos de inpainting (UNET Versión 27)

Segundo Mejor modelo (UNET Versión 29)

Al haber cierta mejoría en los resultados de validación de la versión 27 frente a la versión 26 pero aumentar significativamente el tiempo de entrenamiento, se decidió cambiar de acercamiento para mejorar nuestro modelo, en vez de aumentar el número de capas UNET se decidió separar la dependencia de la función de activación PReLU y aumentar su número de parámetros, esto quiere decir que cada capa UNET tendrá su propia función de activación adaptable, lo cual le da mayor expresividad a la no linealidad del sistema.

- Resultados:
  - Tiempo promedio de entrenamiento: 33263.7 segundos
  - Error mínimo de entrenamiento: 0.020658850132446713 (id 463)
  - Error mínimo de validación: 0.012914350217393933 (id 463)

- Número de Parámetros:
  - Total de parámetros: 5,871,843
  - Parámetros PReLU: 1,920

Resultados de métricas de calidad de imagen (UNET Versión 29)

Ejecución	Error L1	Error L2	PSNR (dB)	SSIM	FID (imagen completa)	FID (más-cara)
Corrida 1	0.022172	0.002720	31.7258	0.9522	1.3528	12.6300
Corrida 2	0.022186	0.002735	31.7035	0.9522	1.3648	12.5704
Corrida 3	0.022216	0.002743	31.6901	0.9521	1.3472	12.7403
Corrida 4	0.022187	0.002724	31.7189	0.9521	1.3528	12.7032
Corrida 5	0.022217	0.002748	31.6825	0.9521	1.3589	12.6043
Corrida 6	0.022155	0.002710	31.7397	0.9522	1.3469	12.5672
Corrida 7	0.022229	0.002739	31.6967	0.9520	1.3576	12.7130
Corrida 8	0.022190	0.002725	31.7147	0.9521	1.3578	12.5805
Corrida 9	0.022216	0.002741	31.6954	0.9521	1.3619	12.6312
Corrida 10	0.022179	0.002715	31.7313	0.9522	1.3549	12.6036
Media ± Std	**0.022195 ± 0.000023**	**0.002730 ± 0.000012**	**31.7099 ± 0.0181**	**0.9521 ± 0.0001**	**1.3556 ± 0.0055**	**12.6344 ± 0.0596**

Tabla 20: Estadísticas de evaluación del modelo best_model_NONBLIND_V29_UNET_NON_SHARED_PRELU7B_MIX_LOSS (10 ejecuciones).

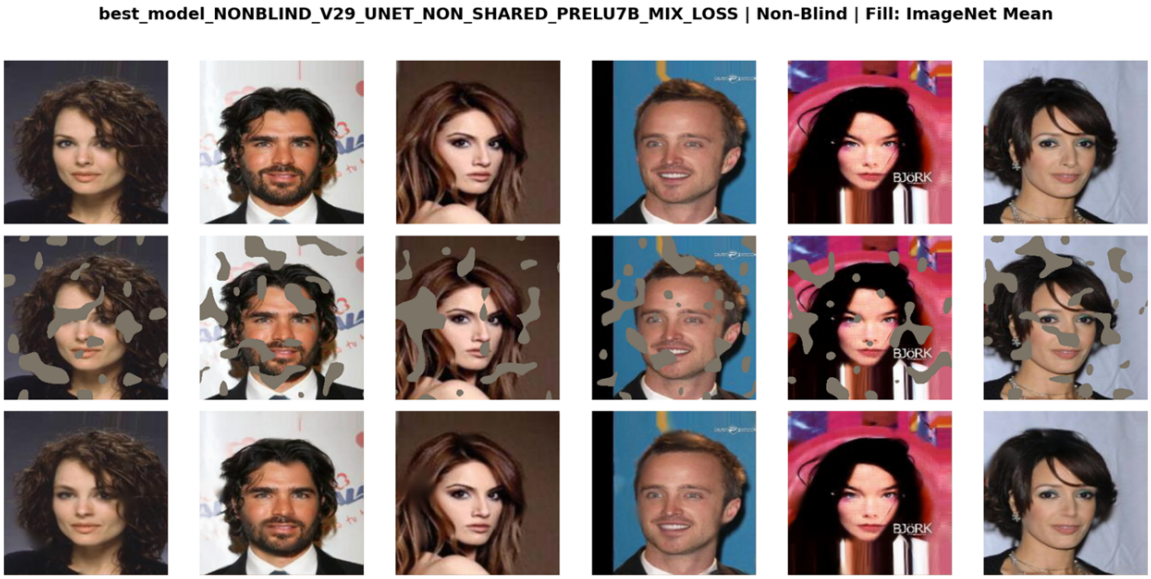


Figura 14: Resultados cualitativos de inpainting (UNET Versión 29)

Mejor modelo (UNET Versión 28)

Sorprendentemente la versión 28 en la cual se usó la arquitectura de 6 capas UNET combinada con la función de activación PReLU con ajuste independiente para cada capa, no solo obtuvo mejores resultados de entrenamiento y validación, sino que fue mucho más estable para el mismo tamaño de imágenes de entrada (256x256). Este modelo, a diferencia de la versión 29 no sufrió de parada temprana y siempre llegó a entrenarse durante 50 épocas. La selección de esta versión también tomó en cuenta las métricas de calidad de imagen y la inspección visual de las reconstrucciones, como se resume en la tabla 23.

- Resultados:
  - Tiempo promedio de entrenamiento: 31626.8 segundos
  - Error mínimo de entrenamiento: 0.020449665153205815 (id 463)

  - Error mínimo de validación: 0.012786873241952377 (id 463)

- Número de Parámetros:
  - Total de parámetros: 4,469,283
  - Parámetros PReLU: 1,536

Resultados de métricas de calidad de imagen (UNET Versión 28)

Ejecución	Error L1	Error L2	PSNR (dB)	SSIM	FID (imagen completa)	FID (más-cara)
Corrida 1	0.022234	0.002668	31.8102	0.9522	1.3534	12.5209
Corrida 2	0.022192	0.002655	31.8323	0.9524	1.3545	12.5583
Corrida 3	0.022206	0.002656	31.8314	0.9523	1.3563	12.6715
Corrida 4	0.022198	0.002655	31.8299	0.9523	1.3479	12.5904
Corrida 5	0.022225	0.002662	31.8188	0.9522	1.3467	12.4826
Corrida 6	0.022208	0.002659	31.8228	0.9523	1.3491	12.7189
Corrida 7	0.022255	0.002682	31.7889	0.9522	1.3543	12.4709
Corrida 8	0.022210	0.002659	31.8257	0.9523	1.3370	12.6109
Corrida 9	0.022211	0.002668	31.8114	0.9523	1.3376	12.4133
Corrida 10	0.022249	0.002673	31.8039	0.9522	1.3448	12.6500
Media ± Std	**0.022219 ± 0.000020**	**0.002664 ± 0.000009**	**31.8175 ± 0.0133**	**0.9523 ± 0.0001**	**1.3482 ± 0.0065**	**12.5688 ± 0.0925**

Tabla 21: Estadísticas de evaluación del modelo best_model_NONBLIND_V28_UNET_NON_SHARED_PRELU6B_MIX_LOSS (10 ejecuciones).

Adicionalmente, para la versión final del modelo se calculó LPIPS utilizando AlexNet sobre aproximadamente el 25% del conjunto de validación de

CelebA. Esta evaluación se registró tanto para la imagen completa como para la región reconstruida, con el objetivo de complementar las métricas basadas en píxeles y distribución de características.

Ejecución	LPIPS (imagen completa)	LPIPS (región reconstruida)
Corrida 1	0.044028	0.031402
Corrida 2	0.043988	0.031388
Corrida 3	0.043848	0.031212
Corrida 4	0.044036	0.031394
Corrida 5	0.043749	0.031128
Corrida 6	0.043925	0.031297
Corrida 7	0.043861	0.031213
Corrida 8	0.043903	0.031312
Corrida 9	0.043892	0.031237
Corrida 10	0.044016	0.031385
Media ± Std	**0.043925 ± 0.000088**	**0.031297 ± 0.000091**

Tabla 22: Resultados LPIPS del modelo best_model_NONBLIND_V28_UNET_NON_SHARED_PRELU6B_MIX_LOSS calculados con AlexNet sobre el 25% del conjunto de validación.

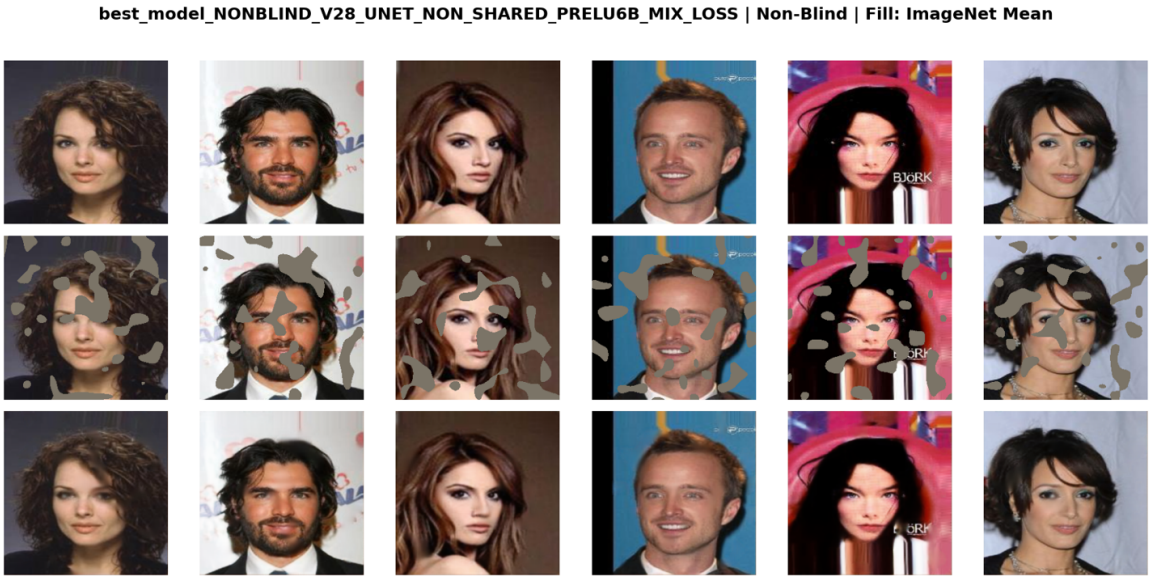


Figura 15: Resultados cualitativos de inpainting (UNET Versión 28)

Comparación final y selección del modelo

Para comparar las cuatro versiones de forma conjunta, la tabla 23 reúne las medias y desviaciones estándar de las métricas calculadas en diez evaluaciones de cada modelo. Para ello, se realizó inferencia sobre las imágenes del conjunto de validación, utilizando diferentes máscaras aleatorias en cada evaluación.

Métrica	V26 192×192	V27 256×256	V28 256×256	V29 256×256
L1 ↓	0.030075 ± 0.000019	0.024951 ± 0.000024	0.022219 ± 0.000020	**0.022195** ± **0.000023**
L2 ↓	0.004298 ± 0.000012	0.003046 ± 0.000009	**0.002664** ± **0.000009**	0.002730 ± 0.000012
PSNR (dB) ↑	29.7400 ± 0.0110	31.2307 ± 0.0126	**31.8175** ± **0.0133**	31.7099 ± 0.0181
SSIM ↑	0.9349 ± 0.0001	0.9450 ± 0.0001	**0.9523** ± **0.0001**	0.9521 ± 0.0001
FID (imagen completa) ↓	2.6391 ± 0.0079	1.6626 ± 0.0096	**1.3482** ± **0.0065**	1.3556 ± 0.0055
FID (máscara) ↓	18.7860 ± 0.1072	13.4488 ± 0.0669	**12.5688** ± **0.0925**	12.6344 ± 0.0596

Tabla 23: Comparación final de métricas de calidad de imagen de las versiones V26, V27, V28 y V29 (media ± desviación estándar). Los encabezados indican el tamaño de entrada descrito para cada versión.

Las negritas indican la media más favorable de cada métrica y las flechas señalan si se busca un valor menor o mayor. LPIPS no se incluye porque se calculó únicamente para V28.

Como puede apreciarse en la tabla, V28 obtuvo las mejores medias en L2, PSNR, SSIM y en las dos variantes de FID. V29 presentó un error L1 ligeramente menor, por lo cual la elección de V28 no se basó en que fuera superior en todas las métricas, sino en valorar estos resultados junto con la calidad visual de las reconstrucciones.

Además, V28 utiliza 4,469,283 parámetros, solo 1,535 más que V26 y

menos que las versiones 27 y 29, con 5,869,924 y 5,871,843 respectivamente. El tiempo promedio de entrenamiento también fue menor que el de estas dos últimas versiones. Por lo anterior, se consideró que V28 ofrecía un balance adecuado entre la calidad de las reconstrucciones y el costo computacional para los objetivos del proyecto.

3.4.6 Comparación de métricas de calidad de imagen con el estado del arte

Es importante señalar que los resultados presentados en esta sección deben interpretarse como una referencia contextual y no como una comparación experimental directa. Los trabajos incluidos en el estado del arte utilizan diferentes conjuntos de datos, resoluciones de entrada, tipos de máscara, porcentajes de área faltante, estrategias de preprocesamiento, arquitecturas y protocolos de evaluación. Por este motivo, el obtener un valor mayor o menor en una misma métrica no demuestra por sí solo que un modelo supera a otro. Para realizar una comparación directa sería necesario evaluar los modelos con las mismas imágenes, máscaras, resoluciones y procedimientos de cálculo. En el presente proyecto no se realizó esta evaluación conjunta, por lo cual los valores de la literatura se utilizan únicamente como referencia. Aun así, esta comparación resulta valiosa para ubicar de manera aproximada el desempeño de nuestro modelo frente a resultados representativos preexistentes en la literatura.

Métrica	Zhou et al. (2020)	Agarwal et al. (2023)	Suvorov et al. (2021)	Liu et al. (2018)	Ivanov et al. (2019)	UNET V28 (Nuestro)	Diferencia %
L1 ↓	0.0146	0.0168	X	**0.0047**	X	0.022219	+372.74%
L2 ↓	X	X	X	X	X	**0.002664**	N/A
PSNR ↑	26.36	28.1843	X	**34.34**	29.3	31.8175	-7.35%
SSIM ↑	0.9107	0.9390	X	0.946	X	**0.9523**	+0.67%
FID (imagen completa) ↓	X	2.012	7.26	X	X	**1.3482**	-32.99%
FID (regiones enmas-caradas) ↓	X	X	X	X	X	**12.5688**	N/A
LPIPS (im-agen com-pleta) ↓	0.0706	X	0.085 / 0.098	X	X	**0.043925**	-37.78%

Tabla 24: Referencia contextual y comparación aproximada de métricas con el estado del arte.

Los valores de la tabla corresponden a las siguientes condiciones reportadas por los autores:

- Zhou et al. utilizan CelebA-HQ a una resolución de 256×256, con máscaras rectangulares aleatorias que cubren entre 13.5% y 25% de la imagen.
- Agarwal y Bhatnagar evalúan Gated Convolution en MaskedFace-CelebA-HQ a 256×256, con oclusiones que simulan un cubrebocas.
- Suvorov et al. reportan resultados de LaMa en CelebA-HQ a 256×256. El FID de 7.26 corresponde a máscaras estrechas, mientras que los

valores de LPIPS de 0.085 y 0.098 corresponden a máscaras estrechas y anchas, respectivamente.

- Liu et al. reportan estas métricas en Places2 a 512×512, con máscaras irregulares que cubren entre 1% y 10% de la imagen. Los valores de L1 y PSNR corresponden a máscaras que pueden tocar los bordes, mientras que el SSIM de 0.946 corresponde a máscaras que no los tocan.
- En uno de sus experimentos, Ivanov et al. obtienen el PSNR de 29.3 en CelebA a 64×64, eliminando aleatoriamente el 80% de los píxeles de forma independiente. Este tipo de enmascaramiento es más cercano a nuestras máscaras tipo MCAR que a las tipo Pattern.

En nuestro caso se utiliza CelebA a 256×256, con máscaras aleatorias tipo Pattern que cubren aproximadamente el 15% de la imagen y los valores mostrados en la tabla corresponden a los promedios de las evaluaciones descritas anteriormente. Para LPIPS se presenta el valor calculado sobre la imagen completa con AlexNet, mientras que el valor sobre la región reconstruida se conserva como una métrica interna adicional. Las celdas marcadas con X indican que no existe un valor reportado para esa métrica y condición en la literatura consultada.

3.4.7 Conclusiones del análisis con métricas de calidad de imagen y ajuste fino

La creación de un flujo de trabajo independiente para evaluar con métricas de calidad de imagen los modelos entrenados en el presente proyecto abrió las puertas a una comparación aproximada con la literatura. Asimismo, nos permitió analizar sus resultados de forma consistente en un entorno controlado.

El hecho de poder ejecutar múltiples evaluaciones para el mismo modelo

con diferentes máscaras aleatorias en cada iteración nos permite asegurarnos de la estabilidad y robustez de una arquitectura dada frente a la aleatoriedad que implica la tarea de inpainting.

Asimismo, el tener un rango basado en la media +- la desviación estándar es más significativo que ejecutar las evaluaciones una sola vez.

En cuanto a la mejora de nuestro modelo se ha llegado a los límites computacionales que es razonable permitirse para una arquitectura UNET con convoluciones simples, obteniendo resultados tanto cualitativos como de calidad de imagen adecuados para los objetivos del proyecto y que incluso pueden considerarse relativamente competitivos frente a la literatura existente. Los trabajos del estado del arte sirvieron como inspiración y referencia, sin que las diferencias numéricas entre sus métricas y las nuestras necesariamente establezcan la superioridad de un modelo frente a otro.

Es importante resaltar el papel que tienen las funciones de transferencia y de pérdida para todo este proceso, ya que ha habido 3 puntos de inflexión a lo largo del desarrollo de este proyecto, y 2 de ellos se han relacionado directamente con dichas funciones.

Como se mencionaba anteriormente la última gran mejoría a nuestro modelo se ha dado gracias al desacoplamiento de nuestra función de activación PReLU, el separar su valor por capa en vez de tener una función global y el darle mayor expresividad a su cálculo mediante el aumento de parámetros que modifican a cada una ha probado ser incluso más efectivo que el aumentar la profundidad de la arquitectura.

Esto último es importante ya que prueba que puede haber mejoras significativas en el desempeño sin incrementar demasiado el número global de parámetros (Un incremento de tan solo 1,535 parámetros frente a 1,400,641).

Otro hallazgo importante yace en evaluaciones realizadas con tamaño de entrada diferente a aquel con el cual fue entrenado un modelo dado, por

ejemplo, para el caso de la arquitectura UNET versión 26 se intentó realizar una evaluación a tamaño (256x256) lo cual resultó en mejores métricas de calidad de imagen, pero peores resultados cualitativos en la reconstrucción. Por otra parte, al reducir el tamaño de las imágenes a (128x128) tanto los resultados cuantitativos como cualitativos empeoraron:

Ejecución	Error L1	Error L2	PSNR (dB)	SSIM	FID (imagen completa)	FID (más-cara)
Corrida 1	0.025694	0.003278	30.9082	0.9443	1.6072	16.1337
Media ± Std	**0.025694 ± 0.000000**	**0.003278 ± 0.000000**	**30.9082 ± 0.0000**	**0.9443 ± 0.0000**	**1.6072 ± 0.0000**	**16.1337 ± 0.0000**

Tabla 25: Estadísticas de evaluación del modelo v26_WITH_256_INPUT_SIZE (1 ejecución).

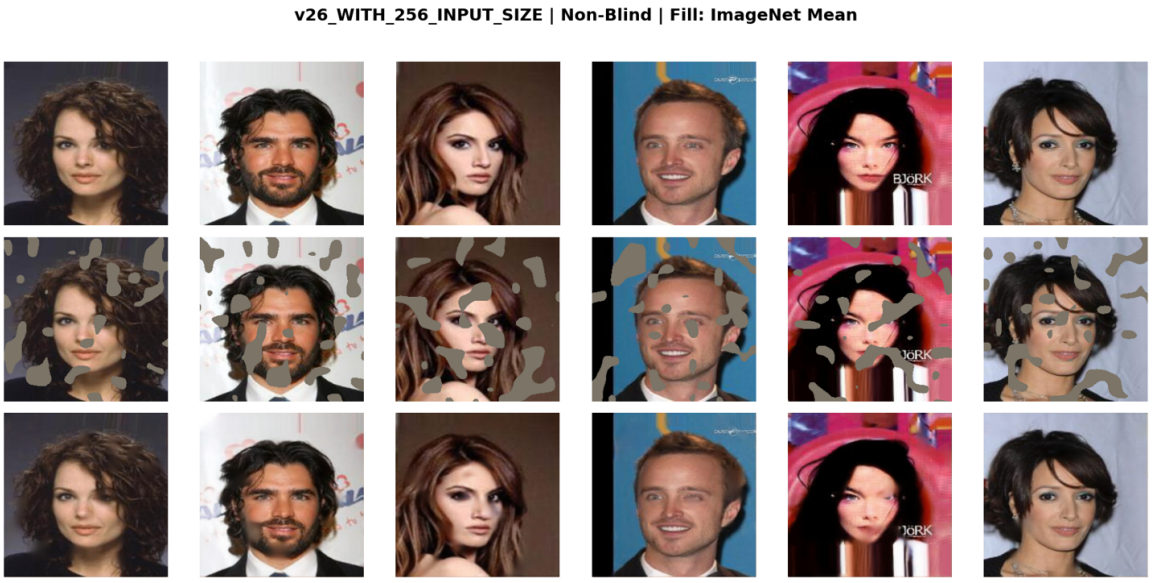


Figura 16: Resultados cualitativos de inpainting con tamaño de entrada mayor al entrenamiento (UNET Versión 26)

Ejecución	Error L1	Error L2	PSNR (dB)	SSIM	FID (imagen completa)	FID (máscara)
Corrida 1	0.040244	0.008152	26.9637	0.9128	6.3835	30.5408
Media ± Std	**0.040244 ± 0.000000**	**0.008152 ± 0.000000**	**26.9637 ± 0.0000**	**0.9128 ± 0.0000**	**6.3835 ± 0.0000**	**30.5408 ± 0.0000**

Tabla 26: Estadísticas de evaluación del modelo v26_WITH_128_INPUT_SIZE (1 ejecución).

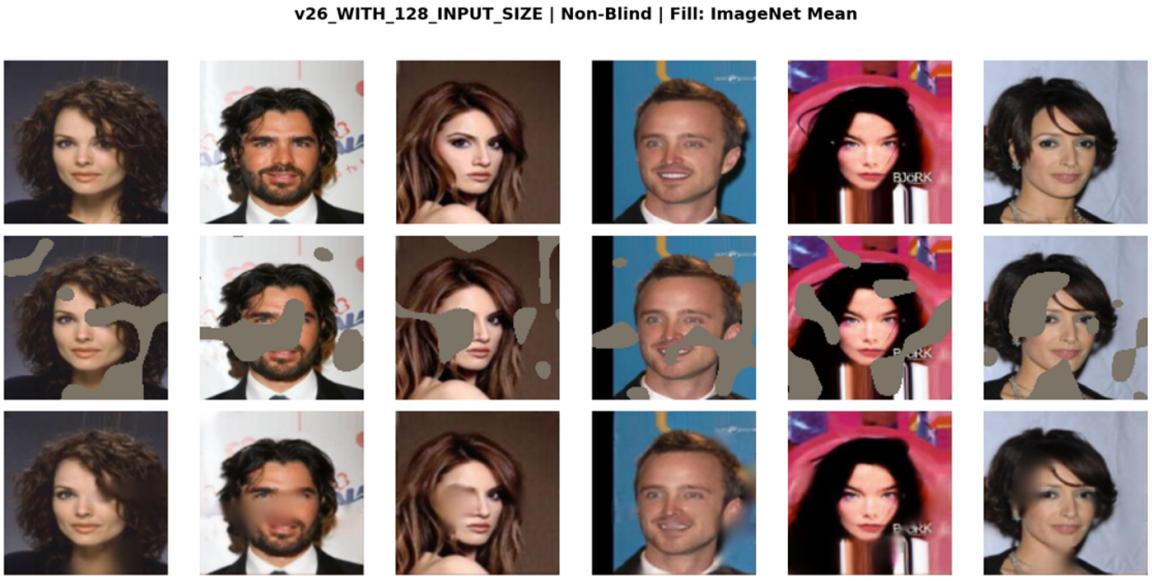


Figura 17: Resultados cualitativos de inpainting con tamaño de entrada menor al entrenamiento (UNET Versión 26)

Este fenómeno ha sido reportado anteriormente en la literatura (Parmar et al., 2021), pero no es discutido frecuentemente en los artículos de inpainting, sin embargo, su descubrimiento nos llevó a dos conclusiones:

1. Es necesario evaluar siempre a la misma resolución a la cual entrenamos.

2. A pesar de ser las más comunes en la literatura, las métricas escogidas pueden resultar inexactas para algunos casos, por ello fue valioso incorporar la similitud perceptual aprendida de parches de imagen (LPIPS) como métrica perceptual adicional. Aunque no todos los trabajos de inpainting la reportan, cuando existe un punto de comparación permite complementar la lectura obtenida mediante L1, L2, PSNR, SSIM y FID.

Para este punto del desarrollo, los resultados tanto cualitativos como cuantitativos de nuestro modelo se consideraron más que suficientes para cumplir con los objetivos planteados para el presente proyecto, la comparación de métricas con la literatura posiciona a nuestra arquitectura V28 como un modelo relativamente competitivo sin ser excesivamente pesado.

Teniendo este número de parámetros fue también razonable considerar la posibilidad de que ese modelo pudiera ejecutarse localmente en equipos de cómputo ligeros como es el caso de las computadoras de placa única (Single-Board Computers, SBCs), lo cual expandiría la utilidad futura de este tipo de desarrollos al no requerir demasiados recursos para ejecutarse en modo de inferencia.

Cabe mencionar, sin embargo, que existe una vertiente de investigación que no pudo ser explorada de manera exitosa durante el desarrollo de este proyecto, y está relacionada con la integración de convoluciones parciales y convoluciones de Fourier a nuestro modelo. Como se mencionó anteriormente el desarrollo de esta arquitectura tuvo 3 puntos de inflexión:

1. La integración de Mix Loss como función de pérdida.
2. El uso de inpainting no ciego, concatenando las máscaras además de aplicarlas a la imagen.
3. El uso de una función de transferencia más expresiva en la forma de una PReLU desacoplada.

Cada uno de estos avances fue incremental y contribuyó por encima del anterior, sin embargo, al intentar aplicar la metodología a las convoluciones parciales y las convoluciones de Fourier nos enfrentamos con la imposibilidad de adaptar estas técnicas sin que esto involucrase comenzar desde cero. Esto se debe a la inflexibilidad inherente de estas arquitecturas, para las cuales la operación de convolución se encuentra ligada estrechamente a las funciones de pérdida establecidas en sus respectivos artículos, lo cual limita su utilidad como una mejora a modelos personalizados y nos forzaría a esencialmente hacer una copia de trabajo existente para poder obtener resultados comparables a los que hemos llegado en el desarrollo de este proyecto. Por lo anterior, estas operaciones de convolución, a pesar de presentarse como una alternativa interesante, tuvieron que ser descartadas.

CONCLUSIONES

Cumpliendo con el objetivo general establecido, en el presente proyecto se logró desarrollar y evaluar un modelo de inpainting facial basado en redes neuronales convolucionales capaz de reconstruir regiones faltantes en imágenes de rostros humanos.

A lo largo de este desarrollo se comprobó que la calidad de las reconstrucciones depende de la combinación de numerosos factores, siendo producto no solo de la arquitectura base elegida, sino también de la optimización de sus parámetros y componentes, entre los cuales destacan: la forma en que se representa la información faltante, la función de pérdida utilizada, las funciones de activación del modelo y la manera en que los resultados son evaluados. Las elecciones finales de cada uno de estos valores son el resultado tanto de rigurosa experimentación iterativa como de la combinación de ideas y recomendaciones presentes en el estado del arte.

Bajo esta perspectiva, el modelo final UNET V28 obtuvo resultados adecuados para los objetivos del proyecto, representando una solución relativamente competitiva con un tamaño de modelo ligero, por lo que servirá como un buen complemento en su futura incorporación a sistemas de retratos hablados.

1. **Elección de la base de datos.**

   El uso de CelebA permitió trabajar con una base de datos suficientemente amplia para el entrenamiento y validación de redes neuronales convolucionales. Al ser esta una base de datos enfocada en rostros humanos, tomarla como base facilitó enormemente el entrenamiento de modelos especializados en reconstrucción facial. A partir de esta base de datos fue posible generar dinámicamente imágenes incom-

pletas mediante distintos tipos de máscaras aleatorias, mientras se mantenía siempre una imagen original como referencia para el cálculo de métricas y funciones de pérdida.

2. **Uso de redes neuronales convolucionales tradicionales para inpainting.**

   Las redes neuronales convolucionales tradicionales demostraron ser una elección adecuada para el problema de inpainting facial, ya que permiten extraer información espacial local y semántica a partir de la imagen de entrada. En particular, las arquitecturas tipo autocodificador convolucional y U-Net resultaron útiles para comprimir la información visual del rostro a sus características más abstractas para posteriormente reconstruir las regiones enmascaradas manteniendo continuidad con las zonas conocidas.

3. **Aplicación del inpainting no ciego.**

   Uno de los puntos de inflexión más importantes de este proyecto fue el cambio de un enfoque de inpainting ciego a no ciego mediante la concatenación de las máscaras como un canal adicional de entrada a las imágenes RGB. Esta modificación permitió que el modelo fuera capaz de distinguir de forma explícita entre las regiones válidas y las regiones faltantes, mejorando tanto los resultados cuantitativos como la coherencia visual cualitativa de las reconstrucciones.

   Al conocer el contexto donde se pretende aplicar este modelo, es más que razonable que las imágenes de entrada del sistema de retratos hablados puedan incluir un canal adicional donde se indique explícitamente la región que se desea reconstruir, siendo este el caso, esta mejora de resultados no sacrifica la utilidad del modelo.

4. **Integración de Mix Loss como función de pérdida.**

   La incorporación de Mix Loss como función de pérdida permitió al modelo combinar la fidelidad píxel a píxel de la pérdida L1 con la

consistencia estructural multiescala de MS-SSIM. Esta combinación resultó especialmente eficaz para el problema de inpainting, ya que permitió mejorar las reconstrucciones generadas sin depender únicamente de una métrica de errores locales o estructuras generales. Este punto de inflexión en el desarrollo favoreció resultados visualmente más coherentes sin aumentar significativamente el tiempo de entrenamiento o inferencia.

5. **PReLU como función de activación.**

   El uso de Parametric ReLU como función de activación permitió introducir una no linealidad más flexible que ReLU, Leaky ReLU o Tanh. Asimismo, el desacoplamiento de sus parámetros aprendidos capa por capa en el modelo final le aportó mayor expresividad al sistema sin incrementar de forma excesiva su número total de parámetros, convirtiéndose en una de las optimizaciones más importantes del modelo y superando incluso a variantes UNET más grandes tanto cuantitativa como cualitativamente.

6. **Validación mediante métricas y comparación con el estado del arte.**

   La evaluación del modelo mediante L1, L2, PSNR, SSIM, FID y LPIPS permitió analizar los resultados de los modelos entrenados desde distintas perspectivas: error píxel a píxel, similitud estructural, similitud distributiva y similitud perceptual.

   Esta combinación de métricas permitió realizar una comparación más completa con trabajos del estado del arte. Los resultados obtenidos fueron competitivos para la mayoría de las métricas elegidas, e incluso con las diferencias entre bases de datos, resoluciones, máscaras, porcentajes de área faltante y protocolos de evaluación, estos resultados representan una referencia valiosa para futuras evaluaciones y mejoras a este desarrollo.

   Los resultados finales indican que el modelo V28 favorece representa-

ciones cercanas a la percepción humana, y se desempeña de manera menos favorable al evaluarse en comparaciones píxel a píxel, indicando cambios en los detalles de las reconstrucciones que sin embargo no perjudican la composición semántica y estructural de la imagen.

7. **Cantidad de parámetros.**

   El modelo final mantuvo un tamaño moderado de 4,469,283 parámetros aprendidos, de los cuales 1,536 corresponden a PReLU. Esta cantidad de parámetros en el modelo vuelve razonable considerar su uso futuro en dispositivos con recursos limitados, como las computadoras de placa única (SBCs), especialmente en escenarios donde la implementación local pueda ser importante.

8. **Flujo de trabajo desarrollado.**

   Además del modelo final, una contribución importante del proyecto yace en el desarrollo de un flujo de trabajo modular de entrenamiento, evaluación, registro y consulta de resultados para redes neuronales artificiales. Tener una estructura de esta naturaleza facilita la experimentación con nuevas arquitecturas, funciones de pérdida, configuraciones de máscaras y métricas, así como el ajuste fino, por lo que puede ser útil para futuros trabajos de investigación desarrollados por otros estudiantes de la universidad.

9. **Impacto y proyección.**

   El modelo desarrollado tiene relevancia directa para el sistema de retratos hablados que motivó este proyecto. Al ser conectado con dicho sistema, el modelo podría utilizarse para completar de forma automática las regiones vacías entre rasgos faciales principales, generando una imagen más continua y visualmente interpretable que apoye procesos de identificación visual en los campos de la seguridad y de la justicia.

RECOMENDACIONES

A partir de los resultados obtenidos y de las limitaciones identificadas durante el desarrollo del presente proyecto, se proponen las siguientes líneas de investigación para trabajo futuro:

1. **Integración del modelo con el flujo de trabajo de retratos hablados.**

   El siguiente paso natural consiste en conectar el modelo final de inpainting (V28) con el flujo de trabajo del sistema de retratos hablados, de forma que el proceso completo pueda recibir los rasgos faciales principales elegidos por el usuario, generar la máscara correspondiente a las secciones que se desean rellenar, y de esta forma, se pueda producir una reconstrucción facial completa. Esta integración permitirá evaluar el desempeño del modelo en el contexto completo para el cual fue desarrollado.

2. **Evaluación de la inferencia del modelo con recursos de cómputo limitados.**

   Debido a que el modelo final mantiene una cantidad de parámetros razonable, se recomienda probar su ejecución en dispositivos con recursos limitados, como las computadoras de placa única. Esto permitiría determinar la capacidad del modelo para utilizarse en escenarios de inferencia local sin depender necesariamente de una estación de trabajo dedicada. Al combinarse con la primera recomendación podría obtenerse una herramienta informática portátil, no dependiente de conexión a internet y útil para el contexto de comunidades aisladas o de bajos recursos.

3. **Construcción de un conjunto de datos mediante segmentación facial.**

Como mejora al modelo final obtenido, sería valioso emplear un modelo de segmentación de imágenes para generar un conjunto de datos de rostros en el cual las regiones de piel se encuentren enmascaradas, conservando únicamente los rasgos faciales principales. Esta estrategia permitiría aproximar de forma más directa el escenario esperado dentro del sistema de retratos hablados, y al ser el entrenamiento mucho más específico se esperaría que los resultados sean mejores para dicha tarea especializada. Un buen punto de partida para esta línea de investigación es la base de datos CelebAMask-HQ, introducida por Lee et al. (2020), ya que proporciona imágenes faciales acompañadas de máscaras semánticas segmentadas, aunque otros modelos de segmentación facial dedicados no quedan descartados.

4. **Experimentación con otras funciones de activación.**

   Aunque PReLU mostró resultados favorables en el modelo final, futuras iteraciones podrían explorar funciones de activación como GELU, ELU o SiLU. La comparación de estas alternativas permitiría determinar si otras funciones de transferencia son capaces de mejorar la estabilidad del entrenamiento, la calidad de reconstrucción o la eficiencia del modelo sin aumentar de forma considerable el número de parámetros.

5. **Uso de pérdidas adversariales.**

   Otra posible línea de mejora consiste en combinar la función de pérdida actual Mix Loss con una pérdida adversarial, de manera similar a los enfoques basados en GAN utilizados en otros trabajos de inpainting. Este tipo de pérdida híbrida podría ayudar a mejorar la nitidez y realismo visual de las reconstrucciones aún más, y al compararse con otros modelos GAN con pérdida meramente generativa adversarial podrían obtenerse descubrimientos interesantes.

REFERENCIAS

1. Hiya Roy, et al. (2021, Abril 6). *Image inpainting using frequency-domain priors*. https://doi.org/10.1117/1.JEI.30.2.023016

2. Omar Elharrouss, et al. (2019, Diciembre 06). *Image Inpainting: A Review*. https://doi.org/10.1007/s11063-019-10163-0

3. Jireh Jam, et al. (2021, Febrero). *A comprehensive review of past and present image inpainting methods*. https://doi.org/10.1016/j.cviu.2020.103147

4. Fernando Berzal. (2018). *Redes Neuronales & Deep Learning*. Editorial Universidad de Granada.

5. Darrow, E. J. (2000). *Pietro Edwards and the restoration of the public pictures of Venice, 1778-1819: necessity introduced these arts*. University of Washington.

6. Elaine Rich. (1983). *Artificial Intelligence*. McGraw-Hill. ISBN 0070522618.

7. Prof. Robert Fullér. (2000). *Introduction to Neuro-Fuzzy Systems*. ISBN 3-7908-1256-0.

8. DeepAI. (s. f.). *Understanding Hidden Layers in Neural Networks*. DeepAI. https://deepai.org/machine-learning-glossary-and-terms/hidden-layer-machine-learning

9. International Business Machines Corporation (IBM). (2021, Agosto 17). *El modelo de redes neuronales*. IBM. https://www.ibm.com/docs/es/spss-modeler/saas?topic=networks-neural-model

10. Castillo, C. (2015). *Algoritmos para la restauración de imágenes digitales en colores*. [Tesis de Maestría. Instituto Politécnico Nacional, Centro de Investigación en Computación]. https://www.saber.cic.ipn.mx/SABERv3/Repositorios/webVerArchivo/25863/1

11. Escalante, B. (2006, Agosto). *Procesamiento Digital de Imágenes*. UNAM. http://lapi.fi-p.unam.mx/wp-content/uploads/PDI_Cap1_Introduccion.pdf

12. Domínguez Torres, Alejandro. (1996, Abril). *Procesamiento digital de imágenes*. ISSN: 0185-2698. https://www.redalyc.org/articulo.oa?id=13207206

13. Marisa Gómez. (2017, Abril). *Historia(s) de la Imagen Digital*. Interartive. https://interartive.org/2017/04/historias-de-la-imagen-digital-marisa-gomez

14. International Business Machines Corporation (IBM). (s. f.). *What is computer vision?* IBM. https://www.ibm.com/topics/computer-vision

15. Farooq Alvi. (2023, Diciembre 13). *Computer Vision and Image Processing: Understanding the Distinction and Interconnection*. OpenCV. https://opencv.org/blog/computer-vision-and-image-processing/

16. International Business Machines Corporation (IBM). (s. f.). *¿Qué es el deep learning?* IBM. https://www.ibm.com/mx-es/topics/deep-learning

17. DataScientest. (2023, Septiembre 10). *Convolutional Neural Network: Everything You Need to Know*. https://datascientest.com/en/convolutional-neural-network-everything-you-need-to-know

18. Sumit Saha. (2018, Diciembre 15). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Towards Data Science. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

19. International Business Machines Corporation (IBM). (s. f.). *What are convolutional neural networks?* IBM. https://www.ibm.com/topics/convolutional-neural-networks

20. Caper Hansen. (2022, Julio 21). *Generative adversarial networks explained*. IBM Developer. https://developer.ibm.com/articles/generative-adversarial-networks-explained/

21. Chandni Agarwal, Charul Bhatnagar. (2023, Mayo 29). *Unmasking the potential: evaluating image inpainting techniques for masked face reconstruction*. https://doi.org/10.1007/s11042-023-15807-x

22. Tong Zhou et al. (2020, Marzo 31). *Learning Oracle Attention for High-fidelity Face Completion*. https://doi.org/10.48550/arXiv.2003.13903

23. NM Salem et al. (2021, Marzo). *A Novel Face Inpainting Approach Based on Guided Deep Learning*. DOI: 10.1109/ICCSPA49915.2021.9385734.

24. Youngjoo Jo, Jongyoul Park. (2019, Febrero 18). *SC-FEGAN: Face Editing Generative Adversarial Network with User's Sketch and Color*. https://doi.org/10.48550/arXiv.1902.06838

25. International Business Machines Corporation (IBM). (2024, Mayo 7). *What is data augmentation?* J. Murel, Ph.D., & E. Kavlakoglu (Contributors). IBM. https://www.ibm.com/topics/data-augmentation

26. International Business Machines Corporation (IBM). (2023, Noviembre 23). *What is an autoencoder?* D. Bergmann & C. Stryker (Contributors). IBM. https://www.ibm.com/topics/autoencoder

27. Ivanov, O., Figurnov, M., & Vetrov, D. (2019). *Variational Autoencoder with Arbitrary Conditioning*. Proceedings of the International Conference on Learning Representations (ICLR 2019). https://arxiv.org/abs/1806.02382

28. Vincent, P., Larochelle, H., Bengio, Y., & Manzagol, P.-A. (2008). *Extracting and composing robust features with denoising autoencoders*. Proceedings of the 25th International Conference on Machine Learning (ICML), 1096–1103. https://doi.org/10.1145/1390156.1390294

29. Burlin, C., Le Calonnec, Y., & Duperier, L. (2017). *Deep image inpainting*. Stanford CS231n Project Report. https://cs231n.stanford.edu/reports/2017/pdfs/328.pdf

30. Xie, J., Xu, L., & Chen, E. (2012). *Image denoising and inpainting with deep neural networks*. Advances in Neural Information Processing Systems. https://www.researchgate.net/publication/264856243

31. Kuo, T.-Y., Wei, Y.-J., Su, P.-C., & Lin, T.-H. (2022). *Learning-based image damage area detection for old photo recovery*. Sensors, 22(21), 8580. https://doi.org/10.3390/s22218580

32. Liu, G., Reda, F. A., Shih, K. J., Wang, T. C., Tao, A., & Catanzaro, B. (2018). *Image inpainting for irregular holes using partial convolutions*. arXiv preprint arXiv:1804.07723. https://arxiv.org/abs/1804.07723

33. Zhao, H., Gallo, O., Frosio, I., & Kautz, J. (2017). *Loss functions for image restoration with neural networks*. IEEE Transactions on Computational Imaging, 3(1), 47–57. https://doi.org/10.1109/TCI.2016.2644865

34. Suvorov, R., Logacheva, E., Mashikhin, A., Remizova, A., Ashukha, A., Silvestrov, A., Kong, N., Goka, H., Park, K., & Lempitsky, V. (2021). *Resolution-robust large mask inpainting with Fourier convolutions*. arXiv preprint arXiv:2109.07161. https://arxiv.org/abs/2109.07161

35. Sohn, K., Yan, X., & Lee, H. (2015). *Learning structured output representation using deep conditional generative models*. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* (Vol. 28, pp. 3483–3491).

Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf

36. Lee, C.-H., Liu, Z., Wu, L., & Luo, P. (2020). *MaskGAN: Towards Diverse and Interactive Facial Image Manipulation*. arXiv preprint arXiv:1907.11922. https://arxiv.org/abs/1907.11922

37. Bergmann, D., & Stryker, C. (2024, Julio 12). *What is a loss function?* IBM Think. https://www.ibm.com/think/topics/loss-function

38. National Instruments. (2024, Junio 28). *Peak signal-to-noise ratio as an image quality metric*. https://www.ni.com/en/shop/data-acquisition-and-control/add-ons-for-data-acquisition-and-control/what-is-vision-development-module/peak-signal-to-noise-ratio-as-an-image-quality-metric.html

39. National Instruments. (s. f.). *Structural similarity index*. NI Vision Concepts Help. https://www.ni.com/docs/en-US/bundle/ni-vision-concepts-help/page/structural_similarity_index.html

40. MathWorks. (s. f.). *ssim: Structural similarity (SSIM) index for measuring image quality*. https://la.mathworks.com/help/images/ref/ssim.html

41. Lawton, G. (s. f.). *What is Fréchet inception distance (FID)?* TechTarget. https://www.techtarget.com/searchenterpriseai/definition/Frechet-inception-distance-FID

42. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2018). *GANs trained by a two time-scale update rule converge to a local Nash equilibrium*. arXiv. https://arxiv.org/abs/1706.08500

43. Parmar, G., Zhang, R., & Zhu, J.-Y. (2021). *On aliased resizing and surprising subtleties in GAN evaluation*. arXiv preprint arXiv:2104.11222. https://arxiv.org/abs/2104.11222

44. Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. In N. Navab, J. Hornegger, W. M. Wells, & A. F. Frangi (Eds.), *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (Lecture Notes in Computer Science, vol. 9351, pp. 234–241). Springer, Cham. https://doi.org/10.1007/978-3-319-24574-4_28

45. MathWorks. (s. f.). *unet: Create U-Net convolutional neural network for semantic segmentation*. MATLAB Help Center. https://www.mathworks.com/help/vision/ref/unet.html

46. Isola, P., Zhu, J.-Y., Zhou, T., & Efros, A. A. (2017). *Image-to-image translation with conditional adversarial networks*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1125–1134. https://openaccess.thecvf.com/content_cvpr_2017/papers/Isola_Image-To-Image_Translation_With_CVPR_2017_paper.pdf

47. Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., & Huang, T. S. (2018). *Generative image inpainting with contextual attention*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 5505–5514. https://openaccess.thecvf.com/content_cvpr_2018/papers/Yu_Generative_Image_Inpainting_CVPR_2018_paper.pdf

48. Wang, Y., Chen, Y.-C., Tao, X., & Jia, J. (2020). *VCNet: A robust approach to blind image inpainting*. arXiv preprint arXiv:2003.06816. https://arxiv.org/pdf/2003.06816

49. Phutke, S. S., Kulkarni, A., Vipparthi, S. K., & Murala, S. (2023). *Blind image inpainting via omni-dimensional gated attention and wavelet queries*. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 1251–1260. https://openaccess.thecvf.com/content/CVPR2023W/NTIRE/papers/Phu

tke_Blind_Image_Inpainting_via_Omni-Dimensional_Gated_Attention_and_Wavelet_Queries_CVPRW_2023_paper.pdf

50. Venkataramanan, A. K., Wu, C., Bovik, A. C., Katsavounidis, I., & Shahid, Z. (2021). *A Hitchhiker's Guide to Structural Similarity*. IEEE Access, 9, 28872–28896. https://arxiv.org/abs/2101.06354

51. MathWorks. (s. f.). *tansig: Hyperbolic tangent sigmoid transfer function*. MATLAB Help Center. https://www.mathworks.com/help/deeplearning/ref/tansig.html

52. Cybenko, G. (1989). *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals and Systems, 2, 303–314. https://doi.org/10.1007/BF02551274

53. Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). *Rectifier nonlinearities improve neural network acoustic models*. Proceedings of the 30th International Conference on Machine Learning. https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf

54. Nair, V., & Hinton, G. E. (2010). *Rectified linear units improve restricted Boltzmann machines*. Proceedings of the 27th International Conference on Machine Learning. https://www.cs.toronto.edu/~fritz/absps/reluICML.pdf

55. He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification*. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 1026–1034. https://www.cv-foundation.org/openaccess/content_iccv_2015/papers/He_Delving_Deep_into_ICCV_2015_paper.pdf

56. Pathak, D., Krähenbühl, P., Donahue, J., Darrell, T., & Efros, A. A. (2016). *Context encoders: Feature learning by inpainting*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2536–2544. https://arxiv.org/abs/1604.07379

57. Zhang, R., Isola, P., Efros, A. A., Shechtman, E., & Wang, O. (2018). *The unreasonable effectiveness of deep features as a perceptual metric*. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 586–595. https://arxiv.org/abs/1801.03924. Página del proyecto: https://richzhang.github.io/PerceptualSimilarity/

58. Zhang, R. (s. f.). *PerceptualSimilarity: LPIPS metric*. GitHub repository. https://github.com/richzhang/PerceptualSimilarity

APÉNDICES

Apéndice 1: Archivo environment.yml

```
channels:
  - pytorch
  - nvidia
  - defaults
  - https://repo.anaconda.com/pkgs/main
  - https://repo.anaconda.com/pkgs/r
  - https://repo.anaconda.com/pkgs/msys2
dependencies:
  - blas=1.0
  - brotli-python=1.0.9
  - ca-certificates=2025.2.25
  - certifi=2025.1.31
  - charset-normalizer=3.3.2
  - cuda-cccl=12.8.90
  - cuda-cccl_win-64=12.8.90
  - cuda-cudart=12.1.105
  - cuda-cudart-dev=12.1.105
  - cuda-cupti=12.1.105
  - cuda-libraries=12.1.0
  - cuda-libraries-dev=12.1.0
  - cuda-nvrtc=12.1.105
  - cuda-nvrtc-dev=12.1.105
  - cuda-nvtx=12.1.105
  - cuda-opencl=12.8.90
  - cuda-opencl-dev=12.8.90
  - cuda-profiler-api=12.8.90
  - cuda-runtime=12.1.0
  - cuda-version=12.8
  - filelock=3.13.1
  - freetype=2.12.1
  - giflib=5.2.2
  - gmp=6.3.0
  - gmpy2=2.2.1
  - idna=3.7
  - intel-openmp=2023.1.0
  - jinja2=3.1.6
  - jpeg=9e
  - khronos-opencl-icd-loader=2024.05.08
  - lcms2=2.16
```

```
- lerc=4.0.0
- libcublas=12.1.0.26
- libcublas-dev=12.1.0.26
- libcufft=11.0.2.4
- libcufft-dev=11.0.2.4
- libcurand=10.3.9.90
- libcurand-dev=10.3.9.90
- libcusolver=11.4.4.55
- libcusolver-dev=11.4.4.55
- libcusparse=12.0.2.55
- libcusparse-dev=12.0.2.55
- libdeflate=1.22
- libjpeg-turbo=2.0.0
- libnpp=12.0.2.50
- libnpp-dev=12.0.2.50
- libnvjitlink=12.1.105
- libnvjitlink-dev=12.1.105
- libnvjpeg=12.1.1.14
- libnvjpeg-dev=12.1.1.14
- libpng=1.6.39
- libtiff=4.5.1
- libuv=1.48.0
- libwebp=1.3.2
- libwebp-base=1.3.2
- lz4-c=1.9.4
- markupsafe=3.0.2
- mkl=2023.1.0
- mkl-service=2.4.0
- mkl_fft=1.3.11
- mkl_random=1.2.8
- mpc=1.3.1
- mpfr=4.2.1
- mpmath=1.3.0
- networkx=3.2.1
- numpy=2.0.1
- numpy-base=2.0.1
- openjpeg=2.5.2
- openssl=3.0.16
- pillow=11.1.0
- pip=25.0
- pysocks=1.7.1
- python=3.9.21
- pytorch=2.5.1
- pytorch-cuda=12.1
- pytorch-mutex=1.0
- pyyaml=6.0.2
- requests=2.32.3
- setuptools=75.8.0
- sqlite=3.45.3
```

```
- tbb=2021.8.0
- typing_extensions=4.12.2
- urllib3=2.3.0
- vc=14.42
- vs2015_runtime=14.42.34433
- wheel=0.45.1
- win_inet_pton=1.1.0
- xz=5.6.4
- yaml=0.2.5
- zlib=1.2.13
- zstd=1.5.6
- pip:
    - adabelief-pytorch==0.2.1
    - beautifulsoup4==4.13.3
    - cachetools==5.5.2
    - cffi==1.17.1
    - colorama==0.4.6
    - contourpy==1.3.0
    - cryptography==43.0.3
    - cycler==0.12.1
    - dnspython==2.7.0
    - fonttools==4.56.0
    - fsspec==2025.3.0
    - gdown==5.2.0
    - google-api-core==2.24.2
    - google-api-python-client==2.166.0
    - google-auth==2.38.0
    - google-auth-httplib2==0.2.0
    - googleapis-common-protos==1.69.2
    - httplib2==0.22.0
    - importlib-resources==6.5.2
    - kiwisolver==1.4.7
    - loguru==0.7.3
    - matplotlib==3.9.4
    - oauth2client==4.1.3
    - packaging==24.2
    - pandas==2.2.3
    - pexpect==4.9.0
    - proto-plus==1.26.1
    - protobuf==6.30.2
    - psutil==7.0.0
    - ptyprocess==0.7.0
    - pyasn1==0.6.1
    - pyasn1-modules==0.4.1
    - pycparser==2.22
    - pydrive2==1.21.3
    - pymongo==4.11.3
    - pyopenssl==24.2.1
    - pyparsing==3.2.1
```

```
  - python-dateutil==2.9.0.post0
  - pytorch-msssim==1.0.0
  - pytorch-ranger==0.1.1
  - pytz==2025.1
  - pywin32==310
  - rclone==0.4.4
  - rsa==4.9
  - six==1.17.0
  - soupsieve==2.6
  - sympy==1.13.1
  - tabulate==0.9.0
  - torch-fidelity==0.3.0
  - torch-optimizer==0.3.0
  - torchaudio==2.5.1
  - torchvision==0.20.1
  - tqdm==4.67.1
  - tzdata==2025.1
  - uritemplate==4.1.1
  - wexpect==4.0.0
  - win32-setctime==1.2.0
  - zipp==3.21.0
```

ANEXOS

Anexo 1: Participación en el Congreso Nacional de Física



SOCIEDAD MEXICANA DE FÍSICA

A quien corresponda.

Tenemos el gusto de informarle que el trabajo con clave única: LXVIII-010373 y cuyo título es: Deep learning aplicado a inpainting en retratos hablados, de los autores:

1. José Marcelo Frías Castro (Presentador)
   Centro de Investigación en Ingeniería y Ciencias Aplicadas
2. Roberto Alan Beltran Vargas
   Centro de Investigación en Ingeniería y Ciencias Aplicadas
3. Braulio Misael Villegas Martínez
   Centro de Investigación en Ingeniería y Ciencias Aplicadas
4. Omar Palillero Sandoval
   Centro de Investigación en Ingeniería y Ciencias Aplicadas
5. José Alfredo Hernández Pérez
   Centro de Investigación en Ingeniería y Ciencias Aplicadas
6. Jorge Arturo Sandoval Espino
   Centro de Investigación en Ingeniería y Ciencias Aplicadas
7. Pedro Antonio Marquéz Aguilar
   Centro de Investigación en Ingeniería y Ciencias Aplicadas

Fue presentado en formato tipo póster en la sesión de Sistemas no Lineales el día 13 de octubre de 2025 en Centro de Convenciones Estado de México dentro del programa del LXVIII Congreso Nacional de Física.

Se extiende la presente constancia el día 29 de mayo de 2026.

ATENTAMENTE,

Por el Comité Científico:

Dr. Juan Carlos Martínez Orozco
Vicepresidente
Sociedad Mexicana de Física.

www.smf.mx

A.P. 70-348, Coyoacán 04511, CDMX (5255) 56224946/56224848 /SMF.info

CERTIFICADO DE PARTICIPACIÓN

LXVIII CONGRESO NACIONAL DE FÍSICA

XL ENCUENTRO NACIONAL Divulgación Científica

smf SOCIEDAD MEXICANA DE FÍSICA

LA SOCIEDAD MEXICANA DE FÍSICA, A.C.
AGRADECE LA PARTICIPACIÓN DE:
(ESTUDIANTE)

JOSÉ MARCELO FRÍAS CASTRO

CENTRO DE INVESTIGACIÓN EN INGENIERÍA Y CIENCIAS APLICADAS

EN EL LXVIII CONGRESO NACIONAL DE FÍSICA
DEL 12 AL 17 DE OCTUBRE DE 2025,
CENTRO DE CONVENCIONES, EDOMEX
TOLUCA, ESTADO DE MÉXICO

DR. ORACIO NAVARRO CHÁVEZ
PRESIDENTE

cnf.smf.mx

GOBIERNO DEL ESTADO DE MÉXICO | ESTADO DE MÉXICO ¡El poder de servir! | Educación Secretaría de Educación Pública | Ciencia y Tecnología

EDUCACIÓN SECRETARÍA DE EDUCACIÓN, CIENCIA, TECNOLOGÍA E INNOVACIÓN | Universidad Autónoma del Estado de México

Anexo 2: Participación en taller o curso para la retribución social




MORELOS
CCYTEM
MUSEO DE CIENCIAS
EL CONSEJO DE CIENCIA Y TECNOLOGÍA DEL ESTADO DE MORELOS
OTORGA LA PRESENTE
CONSTANCIA
a:
Jose Marcelo Frias Castro
Por su participación como tallerista en el evento "Ruta de la Ciencia"
Cuentepec Temixco, Morelos; 20 de junio de 2025
Jaime Eugenio Arau Roffiel
Director General del Consejo de Ciencia y Tecnología
del Estado de Morelos
Alejandra Ramírez Mendoza
Directora del Centro Morelense de
Comunicación de la Ciencia



LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS ,
A TRAVÉS DEL CENTRO DE INVESTIGACIÓN EN INGENIERÍA Y CIENCIAS APLICADAS

OTORGA LA PRESENTE CONSTANCIA A

JOSÉ MARCELO FRIAS CASTRO

Por su destacada participación como TALLERISTA en la "Noche de Observación Estelar", actividad de divulgación de la ciencia y fomento de las vocaciones científicas, dirigido a público en general. Que se llevó a cabo el pasado 5 de noviembre de 2025, en las instalaciones del Centro de Investigación en Ingeniería y Ciencias Aplicadas.

Atentamente
Por una humanidad Culta

DR. J JESÚS ESCOBEDO ALATORRE
ENCARGADO DE DESPACHO DE LA DIRECCIÓN DEL CENTRO DE INVESTIGACIÓN EN INGENIERÍA Y CIENCIAS APLICADAS DE LA UAEM

Folio: 436

UAEM RECTORÍA 2023-2029

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

Se expide el presente documento con firma electrónica UAEM, soportada por el certificado vigente a la fecha de su elaboración y con efectos plenos de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS PUBLICADOS en el ÓRGANO INFORMATIVO UNIVERSITARIO "ADOLFO MENÉNDEZ SAMARÁ" número 117 de fecha 20 de abril de 2021.

Sello electrónico

J JESUS ESCOBEDO ALATORRE | Fecha:2025-12-01 11:20:21 | FIRMANTE

gWKLHDCi8sc/vJ3ZRiEiKdLAJ4cWsAjPwQpWVEvRMgMMO15NvcDm/CxLgQV9rbSKYUazWeC0bbKR1tY89XxFg1pppCuprt+MiZHGcKN4XknD9LTFNzYvAFCuCqpsuuWRPY+2i7BeCUoCex3ANYVBOu+ZqpgZeroO8Qw9eK4i8BVhNtuFsaeIDQgpKMBfnw3mE1mpyOV1AVPMOdI3QLxZgkWnu2MYkJFM87QQFWvkuBgtQiQHL9tJRPgxJK4djYRnD05W72CoSeWrYTC1jZCcMiCy5wzowmY1XBgBH4yHxTW25vq9aZSfuS885tJBPYXzbFhuz0pasJoyRM8y3dY1Q==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

xNjJYqKk

https://efirma.uaem.mx/noRepudio/1r2pVxH3Z6KYLiZIpJYFYXXgeGhCx7K

UAEM RECTORÍA 2023-2029

Cuernavaca, Morelos, 07 de septiembre de 2026.

DR. ARTURO MOLINA OCAMPO
COORDINADOR DEL POSGRADO
EN INGENIERÍA Y CIENCIAS APLICADAS
P R E S E N T E

Atendiendo a la solicitud para emitir DICTAMEN sobre la revisión de la TESIS titulada: **DEEP LEARNING APLICADO A INPAINTING EN RETRATOS HABLADOS** que presenta el alumno **JOSÉ MARCELO FRÍAS CASTRO**, para obtener el título de la **MAESTRÍA EN INGENIERÍA Y CIENCIAS APLICADAS** .

Nos permitimos informarle que nuestro voto es **APROBATORIO.**

Atentamente
Por una humanidad culta

DR. JOSÉ ALFREDO HERNÁNDEZ PÉREZ
DR. OMAR PALILLERO SANDOVAL
DR. JORGE ARTURO SANDOVAL ESPINO
DR. BRAULIO MISAEL VILLEGAS MARTÍNEZ
DR. ROBERTO ALAN BELTRÁN VARGAS

Se adiciona efirma UAEM

UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS

Se expide el presente documento con firma electrónica UAEM, soportada por el certificado vigente a la fecha de su elaboración y con efectos plenos de conformidad con los LINEAMIENTOS EN MATERIA DE FIRMA ELECTRÓNICA PARA LA UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MORELOS PUBLICADOS en el ÓRGANÓ INFORMATIVO UNIVERSITARIO "ADOLFO MENÉNDEZ SAMARÁ" número 117 de fecha 20 de abril de 2021.

Sello electrónico

OMAR PALILLERO SANDOVAL | Fecha:2026-09-07 12:31:00 | FIRMANTE

mduS5Qp5D/WuR9qzlNR2f+CMK2TcX6x+/ScfRudTJJ1LWXU8WggpmIqXGydcL4HiqiME8RAhXIhmla0lTEXO4RqzZUixvKPXLPZakJhSohfNmHpXthUVzoBO4Q35SpQID+q9F
WGX5we2zoRzSYQm28KavMGPRp1e5PpyfqQLLOQVe4U+QoX5q6evKymgpOh57mqnOTcBdOrxas8azbZfyPbMWjMtprNWhFHgg8IxMqXcYFPKSDlYlcmTu35V2vS4CnM14J
KLy/3b3Lkcu+TM0N5qJiUUhUEy/F5fyS3Bwivhx5bWsfmolcD1yj/4hH8C78BJhz/ZIYOHOJ71wUxO9g==

JOSE ALFREDO HERNANDEZ PEREZ | Fecha:2026-09-07 13:20:23 | FIRMANTE

f0Jw7J43Sbmj8gZX5N2bA25FI8vklR0iVULF9xcduiw7Q2T6/0PdhLYYLrj3haRdAfPLnuLE85SmfDSsU48v2TB+euV6YShtFH8WpEaaSTxWtqO/e7VdHjwjKIszZN1k2UCx5D257k1
eFdLGEDS3e2v8wcDdgtxnGfNTXrBwmLkB91ct6aQOjQYQ9o4xaw0l4qdx8zqLbac0Vd8vFwqyupxpVDureJa2Buw5AcAG58AQfIHZrdRY0x9fwHgmSpQq6Y6iQVWnFj0QDlTz3Y/
FHSfXbG8ShAJ2lWzlMNqw9QVRwTDa5jfsMXWiBwze03dYqZKhJpgIES47XCtXExo8YQ==

JORGE ARTURO SANDOVAL ESPINO | Fecha:2026-09-07 15:01:30 | FIRMANTE

tnx3g6REH1b3TilerWRePtQCUIABAR218ptkJ7PoW67GTnWld/RLwAh/RnJtjl+tWnyesEcaU1EC6oBd4VdigXAw2SYesC3A52F2YMgwtgZrr2omLOJQ5jI9kezU5AgN+3ia4hihmKz
L1CJ6DLh1QSqLPXhzylNjg7AsIiFgkIi2Iv6uEnX3w3+etbceMp0VNdTvhWTmPXZeEzBbR03tZRAw9HT2qfTcABnt+DK8ql71iC/lcnJ/cji9v7ChYO92wp22TpkxXeSxL1avM+kjOz0m
TsXSV0Wa6gdE6gLfG7pVdv+KNEwMsddr77/d6R+N+0ukhFQNxkxKh0NCSWeFYg==

BRAULIO MISAEL VILLEGAS MARTINEZ | Fecha:2026-09-07 15:21:10 | FIRMANTE

SIbXIo3SV2LrxrF91SH3zhpQIgDZOXwkIiI28XSsrW73yOIa9KF1R1Nxen2q2vn99tbVW8poZGKJ3RQegbFthop4GgaXlmto67jPup1kHZXsDUxSxRtsj1p303Wunl+5vL5sLLD8H08
O4/hzkKo6gDGzFbEJuGuvxg52Y1BSQhqZjDuWgjAzYgTbk2dDUOSinu5CvDA208BpSCSNgSgnwMNt+Wj2yA+7cgVW6IodICsD0p82/gi6uUisZP0qoeiDtcm4HQEVVRScqXQa7
TfHQF/+tRlIE1othScxUhy5vLrFXryfk4ngDsWJuGyQV/pEK1cCSYA8anpkyQXGDlUUDA==

ROBERTO ALAN BELTRAN VARGAS | Fecha:2026-09-07 20:52:57 | FIRMANTE

X4lxRQcXDA1GKHwWm55fioLXL6xjMojjIkPqqTE1Rk4UgKTWQZGkWvojbeUyhuXMSYdjyj24uk/vuJuikh4uxs9Ia/Wq0P4fMmjuoAfebj4OuNdyFjtnb0nzKJyDX/enpVFA/V1w+Y/9a
kYdoM1BzCRWLIQPl1S5Sww8NxxMcUxiMVyRXi/aAQ4BSv6uxl0QnVjxVyw0OfEAWZ0a7tB9z2uypQ5No7tl9JvNCsy+jG+T/1Mbj75xfkLNW9ySA1ozmfpgsQKSHnLpa/06S2Xdd
RqezDInLIwIQ8k6hz81ub6CR/vxQ+AZfhTJik/YDZJXQYHVeOYlRA3PIPPn/yrilw==

Puede verificar la autenticidad del documento en la siguiente dirección electrónica o escaneando el código QR ingresando la siguiente clave:

OMw9WYXtp

https://efirma.uaem.mx/noRepudio/6m3LQcTgnamnkGzhTJdyh2QdGL7Gtwk0