

UNIVERSIDAD POLITÉCNICA DE MADRID
E.T.S. DE INGENIERÍA DE SISTEMAS INFORMÁTICOS
PROYECTO FIN DE GRADO
GRADO EN CIENCIA DE DATOS E INTELIGENCIA ARTIFICIAL

Generación de prendas de moda mediante IA generativa

Desarrollado por: Patricia Fuxi Alcalde Benítez

Dirigido por: Edgar Talavera Muñoz

Madrid, 17 de julio de 2025

Generación de prendas de moda mediante IA generativa

Desarrollado por: Patricia Fuxi Alcalde Benítez

Dirigido por: Edgar Talavera Muñoz

Proyecto Fin de Grado, 17 de julio de 2025

E.T.S. de Ingeniería de Sistemas Informáticos

Campus Sur UPM, Carretera de Valencia (A-3), km. 7

28031, Madrid, España

Si deseas citar este trabajo, la entrada completa en BIBTEX es la siguiente:

```
@mastersthesis{citekey,  
    title = {Generación de prendas de moda mediante IA generativa},  
    type = {Bachelor's Thesis},  
    author = {Patricia Fuxi Alcalde Benítez},  
    school = {E.T.S. de Ingeniería de Sistemas Informáticos},  
    year = {2025},  
    month = {7},  
}
```

Esta obra está bajo una licencia [Creative Commons «Atribución-NoComercial-CompartirIgual 4.0 Internacional»](#). Obra derivada de <https://github.com/blazaid/UPM-Report-Template>.



Todo cambio respecto a la obra original es responsabilidad exclusiva del presente autor.

Agradecimientos

En primer lugar, agradecer a mis padres por su apoyo constante y su amor incondicional. Gracias por ser un ejemplo a seguir en todos los aspectos de la vida y por inspirarme para llegar a este momento. Sin vuestros ánimos, motivación y palabras de reafirmación todo habría sido mucho más abrumador.

A mi familia por su apoyo y confianza en todos los momentos y por siempre velar por mí.

A Claudia por ser mi lugar seguro. Muchas gracias por todas tus sabidurías, todas las charlas interminables en las que independientemente de nuestro estado siempre conseguían hacernos sentir comprendidas y optimistas.

A mis amigas del instituto por ser una distracción en los peores momentos, por vuestro apoyo, todos los momentos innolvidables que hemos compartido y por enseñarme lo que es la amistad verdadera.

A mis amigas de baile por inspirarme cada día para ser mujeres independientes, por toda compartir toda vuestra experiencia conmigo y por todos los momentos que compartimos haciendo lo que más nos gusta, bailar.

A los amigos que he hecho durante la carrera, tanto en Montegancedo como en el Campus Sur que han hecho que esta etapa universitaria sea mucho más llevadera y disfrutable.

Por último, a mi tutor Edgar por darme la oportunidad de realizar este trabajo y por confiar en mí.

Resumen

El uso de técnicas de inteligencia artificial generativa está en plena expansión, siendo cada vez más común encontrar modelos generativos en ámbitos diversos como el arte o el diseño.

Este trabajo pretende examinar el impacto de estas técnicas aplicadas al diseño de moda y su capacidad de para generar imágenes realistas y significativas. Para ello se ha seguido una metodología estructurada que comienza con la recolección de los datos de la página web *Farfetch* mediante técnicas de *web scraping*. A continuación, se han implementado y evaluado tres modelos: DCGAN, VAE y modelos de difusión. Además, para cada modelo se han desarrollado distintas variaciones arquitectónicas para comparar cómo varían los resultados y se ha analizado el efecto de integrar data augmentation a nuestro conjunto de datos. Finalmente se muestran los resultados y gráficas de las funciones de pérdida para comprobar el rendimiento del entrenamiento y la calidad del modelo.

Sin embargo, este trabajo no se limita solo al desarrollo y evaluación de los modelos y resultados. También se plantean ideas de futuros proyectos y líneas de investigación, se reflexiona sobre su aspecto social y medioambiental y presenta debates éticos y filosóficos que surgen en torno a su uso en la industria de la moda y en campos predominantemente creativos.

Palabras clave: Inteligencia Artificial generativa; Generación de imágenes; Redes neuronales; Deep Convolutional Generative Adversarial Network Variational Autoencoders; Modelos de difusión;

Abstract

The use of generative artificial intelligence techniques is in full expansion, being more and more common to find generative models in diverse fields such as art or design.

This paper aims to examine the impact of these techniques applied to fashion design and the ability of these technologies to generate realistic and meaningful images. For this purpose, a structured methodology has been followed, starting with the collection of data from the website *Farfetch* by means of *web scraping* techniques. Then, three models have been implemented and evaluated: DCGAN, VAE and diffusion models. In addition, for each model different architectural variations have been developed to compare how the results vary and the effect of integrating data augmentation to our dataset has been analyzed. Finally, the results and plots of the loss functions are shown to check the training performance and the quality of the model.

However, this work is not limited only to the development and evaluation of the models and results. It also raises ideas for future projects and lines of research, reflects on its social and environmental aspect, and presents ethical and philosophical debates that arise around its use in the fashion industry and in predominantly creative fields.

Keywords: Generative artificial intelligence; Image generation; Neural networks; Deep Convolutional Generative Adversarial Network; Variational Autoencoders; Diffusion models;

Índice general

1	Introducción	1
1.1	Motivación	2
1.2	Objetivos	2
1.3	Estructura de la memoria	3
2	Estado de la cuestión	5
2.1	Redes Neuronales	6
2.2	Modelos generativos	14
2.3	Modelos autoregresivos y variantes recientes	30
2.4	Aplicaciones de IA en la industria de la moda	31
3	Metodología	32
3.1	Descripción General del Enfoque	32
3.2	Dataset	33
3.3	Preprocesamiento	36
3.4	DCGAN con 4 capas convolucionales	38
3.5	DCGAN con 4 capas convolucionales y data augmentation	40
3.6	DCGAN con 6 capas convolucionales y data augmentation	42
3.7	VAE con 3 capas convolucionales con data augmentation	43
3.8	VAE con arquitectura convolucional profunda y data augmentation	45

3.9	Modelo de Difusión	46
4	Resultados	50
4.1	Presentación de los resultados	50
4.2	Comparación global de los modelos	57
4.3	Discusión de los resultados	59
5	Proyectos Futuros	61
6	Impacto	63
6.1	Impacto medioambiental	63
6.2	Impacto social	64
7	Conclusiones	66
A	Repositorio del proyecto	67

Índice de figuras

1.1 Colección del diseñador Kübra Karasu's presentada en la segunda edición de AI Fashion Week. Fuente: University of Fashion	2
2.1 Estructura básica de una Red Neuronal Artificial. Fuente: Wikipedia	9
2.2 Capas de una CNN. Fuente: ResearchGate	12
2.3 Proceso de convolución. Fuente: Medium	12
2.4 Max-pooling. Fuente: Medium	13
2.5 Capa completamente conectada. Fuente: Medium	13
2.6 Ejemplo de la arquitectura de una GAN. Fuente: ResearchGate	17
2.7 Arquitectura del Generador de una DCGAN. Fuente: Medium	21
2.8 Arquitectura del Discriminador de una DCGAN. Fuente: Medium	22
2.9 Arquitectura de una red DCGAN. Fuente: UPM	22
2.10 Arquitectura de un autoencoder. Fuente: ResearchGate	24
2.11 Comparación entre los autoencoders tradicionales con los autoencoders variacionales. Fuente: Manning	26
2.12 Funcionamiento de un modelo de difusión. Fuente: CSDN	28
3.1 Comparación de ambas imágenes	34
3.2 Imágenes del dataset después del preprocessamiento	37
3.3 Ejemplo de imagen sintética generada por el generador	38

3.4	Muestras del nuevo dataset después de aplicar técnicas de data augmentation	41
3.5	Evolución de una imagen durante el proceso de difusión	47
4.1	Imágenes generadas a lo largo de 20 epochs	50
4.2	Gráfica de pérdidas del entrenamiento	50
4.3	Imágenes generadas cada 10 epochs por el modelo DCGAN básico	51
4.4	Gráfica de pérdidas del entrenamiento de la DCGAN con data augmentation	52
4.5	Evolución de las imágenes generadas por este modelo a lo largo de las 100 epochs	53
4.6	Gráfica de pérdidas del entrenamiento del modelo 2 DCGAN	53
4.7	Imágenes generadas por el modelo Variational Autoencoder con una arquitectura básica	54
4.8	Evolución de la métrica ELBO en el primer modelo Variational Autoencoder	54
4.9	Imágenes generadas por el modelo Variational Autoencoder con una arquitectura más compleja	55
4.10	Evolución de la métrica ELBO en un Variational Autoencoder con una arquitectura más compleja	55
4.11	Proceso de generación de imágenes durante T en las epochs 0, 30, 65 y 100	56
4.12	Gráfica de la pérdida del modelo de difusión	56
4.13	Comparativa de resultados entre tres modelos entrenados durante 100 épocas, con una arquitectura simple y empleando data augmentation	58
4.14	Comparativa de resultados entre dos modelos entrenados durante 100 épocas, con una arquitectura convolucional más profunda y empleando data augmentation	59

Índice de tablas

4.1	Pérdidas de las primeras y últimas 3 épocas	50
4.2	Pérdidas: primeras y últimas 3 épocas	52
4.3	Pérdidas: primeras y últimas 3 epochs	53

1.

Introducción

La industria de la moda se encuentra en constante evolución, cada vez más influenciada por las nuevas tecnologías. La tecnología ha sido una parte esencial tanto en el proceso de diseño como en el de producción desde mediados del siglo XVIII.

Todo comenzó con la invención de la máquina de coser durante la Revolución Industrial de los siglos XVIII y XIX. Durante las primeras décadas del siglo XX, surgieron nuevos materiales como la goma y el plástico que permitieron el diseño de nuevos productos textiles. En las últimas décadas del siglo XX, el rápido desarrollo de los ordenadores e Internet ha transformado completamente el mundo de la moda y el diseño. Programas de software como [Computer Aided Design](#) y [Telar de Jacquard](#) han alterado la manera en la que se diseñan y confeccionan los vestidos.

La última revolución digital, la [inteligencia artificial \(IA\)](#), ha transformado numerosos sectores gracias al aprendizaje profundo y, en particular, a los modelos generativos, capaces de crear contenido original como imágenes, texto o audio a partir de datos aprendidos. Estas técnicas se han aplicado con éxito en áreas como el arte digital, el diseño gráfico o el desarrollo de videojuegos, y cada vez más están ganando presencia en el mundo de la moda.

En este contexto, la IA generativa ofrece nuevas posibilidades en el proceso creativo: automatización del diseño, generación de variantes estilísticas, simulación de pasarelas digitales o personalización de prendas. Entre las técnicas más utilizadas destacan [Generative Adversarial Network \(GAN\)](#), los [Variational Autoencoders \(VAE\)](#) y los [modelos de difusión](#), cada una con ventajas particulares en la generación de imágenes realistas, estructuradas o progresivas.

Este trabajo se centra en el estudio y comparación de estas tres técnicas aplicadas al diseño de prendas de moda. Concretamente, se desarrollan y entrena diferentes variantes de las tres tecnologías mencionadas anteriormente sobre un conjunto de datos construido a partir de artículos de marcas de lujo, con el fin de evaluar su viabilidad como herramientas creativas. Además, se analiza el impacto de técnicas de aumento de datos en el entrenamiento y se reflexiona sobre el papel de la IA generativa en la industria de la moda desde una perspectiva ética, social y medioambiental.

1.1. Motivación

Modelos de IA generativa como las GAN o los modelos de difusión han demostrado su capacidad para crear imágenes visualmente convincentes. Estas tecnologías están empezando a desempeñar un papel importante en los procesos creativos, permitiendo la generación automatizada de diseños, la exploración de nuevas estéticas visuales y la personalización a gran escala.

Esta técnica facilita la generación de diseños de alta calidad, mejora la eficiencia del proceso y permite la creación de nuevas prendas tanto funcionales como visualmente estéticas. Aunque su uso no está completamente extendido, especialmente en el diseño de colecciones para las pasarelas, cada vez son más los diseñadores que recurren a estas herramientas para desarrollar y presentar sus colecciones ya que les permite, por un lado, ahorrar tiempo (al evitar la confección manual de los prototipos) y dinero (al no requerir materiales físicos). Esta tendencia va en aumento y, de hecho, en el 2023 se añadió al calendario una nueva semana de la moda dedicada exclusivamente a colecciones generadas mediante [inteligencia artificial](#), la AI FASHION Week (Figura 1.1).



Figura 1.1. Colección del diseñador Kübra Karasu's presentada en la segunda edición de AI Fashion Week. Fuente: [University of Fashion](#)

1.2. Objetivos

En este contexto, el presente trabajo se enmarca en la investigación de la aplicación de la IA generativa al diseño de moda, analizando su potencial como herramienta creativa.

Este trabajo se centra en el uso de modelos generativos para la creación de prendas de

ropa. Concretamente, se analizan tres enfoques representativos del estado del arte: las **GAN**, los **VAE** y los **modelos de difusión**. Estas tres técnicas han sido ampliamente utilizadas en tareas de generación de imágenes por su capacidad para aprender distribuciones complejas de datos visuales. Las **GAN** son conocidas por su realismo visual, aunque difíciles de entrenar; los **GAN** permiten estructurar el espacio latente de forma útil para manipulación y compresión; y los **modelos de difusión** han demostrado ser especialmente eficaces en la generación progresiva de alta calidad.

A través del entrenamiento y la evaluación de diferentes técnicas generativas sobre un conjunto de imágenes de prendas, se busca analizar la viabilidad y la capacidad de dichas técnicas para crear prendas visualmente coherentes y atractivas con el fin de explorar su utilidad en el proceso creativo del diseño de moda.

Para alcanzar este propósito, se han fijado los siguientes objetivos generales con los cuales este proyecto cumplirá:

- Entrenamiento, evaluación y comparación de tres modelos generativos implementados en el lenguaje de programación python, sobre un conjunto de datos que recoge imágenes de artículos de marcas de lujo durante el mes de abril de 2025.
- Comprender las limitaciones presentes durante el desarrollo y proponer mejoras en el rendimiento de los modelos entrenados.
- Analizar la coherencia, variedad, calidad visual y originalidad resultados y entender hasta qué punto es posible generar contenido visual original en el contexto actual.
- Reflexionar sobre el papel de la **IA** en la creación artística, su impacto futuro en la industria y los desafíos y posibles dilemas éticos que puede plantear.

1.3. Estructura de la memoria

La memoria del presente trabajo se estructura de la siguiente forma:

1. **Introducción:** la introducción es la parte introductoria en la que se contextualiza el tema del trabajo, los objetivos marcados y la motivación.

2. **Estado de la cuestión:** el estado de la cuestión es una sección fundamental en cualquier estudio científico ya que sirve para introducir las técnicas existentes y enfoques en el ámbito de investigación del proyecto.
3. **Marco teórico:** esta sección sirve para explicar los aspectos teóricos de los modelos utilizados con el fin de facilitar posteriormente la comprensión de la implementación y funcionamientos de los mismos.
4. **Metodología:** se desarrolla detalladamente el proceso seguido durante este trabajo. Desde la obtención de los datos hasta la implementación y arquitectura de los modelos utilizados.
5. **Resultados:** en esta sección se muestran y se comparan los resultados obtenidos de cada una de las técnicas implementadas. Además, se incluye una discusión de los mismos para recalcar los puntos fuertes y débiles de cada modelo.
6. **Proyectos Futuros:** este capítulo propone nuevas líneas de investigación para el futuro dentro del área.
7. **Impacto:** aquí no solo se presenta el impacto medioambiental de este trabajo sino que también se analizan las implicaciones sociales que surgen.
8. **Conclusión:** esta última sección engloba una síntesis del proyecto completo, nombrando las aportaciones más significativas y ofreciendo una visión global sobre el impacto del proyecto.

2.

Estado de la cuestión

La [inteligencia artificial](#) generativa es el conjunto de técnicas capaz de crear contenido original (imágenes, vídeos, texto, audio o código) a partir de modelos de aprendizaje profundo, algoritmos que simulan los procesos de aprendizaje y toma de decisiones del cerebro humano. Estos modelos aprenden a capturar las relaciones, estructuras y patrones latentes en conjuntos de datos; y utilizarlos para generar nuevas muestras realistas y coherentes.

En los últimos años, esta tecnología ha experimentado una gran popularidad, especialmente impulsada en gran parte por el éxito de modelos de lenguaje como ChatGPT; provocando un aumento en la innovación y adopción de la [IA](#) en ámbitos como la pintura, el arte, la literatura y, por supuesto, la moda y el diseño.

En términos generales, el proceso de desarrollo de un modelo generativo se puede dividir en tres etapas fundamentales:

- **Entrenamiento:** para alimentar un modelo fundacional con grandes volúmenes de datos para que aprenda representaciones latentes y estructuras generales del dominio, en este caso, moda.
- **Ajuste:** una vez entrenado el modelo base, se puede ajustar para tareas o estilos específicos, utilizando subconjuntos más pequeños de datos más especializados.
- **Generación, evaluación y reajustes:** para evaluar la salida y mejorar su calidad y precisión.

Para comprender con mayor facilidad el desarrollo de los modelos y las arquitecturas utilizadas se presentarán las técnicas básicas en [inteligencia artificial](#) que forman la base de los modelos generativos existentes.

2.1. Redes Neuronales

Una red neuronal artificial es un modelo de *Machine Learning* que busca imitar el comportamiento biológico del cerebro humano y las conexiones entre las neuronas. Una red neuronal artificial está compuesta por muchas unidades funcionales simples, también llamadas neuronas artificiales, que procesan información con el objetivo de resolver tareas de clasificación o regresión. Las neuronas se agrupan en diferentes capas y se interconectan. De esta forma, el modelo puede interactuar con el entorno mediante capas de neuronas tanto para recibir información como para transmitirla. Además, el sistema puede transferir información entre capas siguiendo ciertas reglas de aprendizaje.

Componentes de una Red Neuronal

La arquitectura de una red neuronal está compuesta por unidades individuales llamadas **neuronas artificiales**, que simulan las neuronas cerebrales. A continuación, se describen los componentes principales de una neurona artificial y cómo se agrupan en capas dentro de la red.

- **Neurona:** componente básico de la red neuronal. Reciben entradas, realizan cálculos y generan salidas. Cada neurona está conectada a otras neuronas mediante conexiones ponderadas. Estas ponderaciones determinan la fuerza de la conexión y desempeñan un papel importante en el proceso de aprendizaje.
- **Capas:** las neuronas se organizan en capas. Cada red neuronal está compuesta por tres capas. La capa de entrada recibe los datos de entrada, la capa de salida produce la salida final y las capas ocultas se encuentran entre ambas y se encargan de aprender los patrones. La profundidad de una red neuronal se refiere al número de capas ocultas que contiene.
- **Peso (Weight):** su función principal es asignar mayor o menor importancia a las diferentes entradas. Se implementa como un producto escalar entre la entrada y una matriz de pesos. Por ejemplo, en un modelo de análisis de sentimiento, una palabra con carga negativa podría tener un peso mayor que una palabra neutral.
- **Sesgo (Bias):** actúa como un término constante que desplaza el valor producido por la función de activación. Su rol es similar al término independiente en una función lineal, ayudando a ajustar mejor la salida del modelo.

Capas de una Red Neuronal

- **Entrada (Input):** la entrada es el conjunto de características (features) que se introduce en la red neuronal para iniciar el proceso de aprendizaje. Por ejemplo, en tareas de visión por computador, esta entrada suele ser un arreglo de valores que representan los píxeles de una imagen.

Una vez que los datos llegan a una neurona, se procesan mediante la función de transferencia. Su función es combinar las múltiples entradas en un único valor de salida, normalmente a través de una suma ponderada de las entradas. Esta operación lineal representa la parte determinista del cálculo en la neurona.

Sin embargo, para que la red neuronal pueda aprender relaciones complejas y no simplemente una combinación lineal, se utiliza una **función de activación**. Esta se aplica inmediatamente después del cálculo lineal en cada neurona y tiene la función de introducir **no linealidad** al modelo. Sin ella, la red sería incapaz de aproximar funciones no lineales, lo cual limitaría severamente su capacidad de generalización. Ejemplos comunes de funciones de activación son ReLU, sigmoid y tanh, cada una con sus propias ventajas dependiendo del tipo de problema y arquitectura.

- **Capa(s) oculta(s):** las capas ocultas son las que definen el aprendizaje profundo como lo es hoy en día. Son capas intermedias que realizan todos los cálculos y extraen las características de los datos.

Puede haber múltiples capas ocultas interconectadas que se encargan de buscar diferentes características ocultas en los datos. Por ejemplo, en el procesamiento de imágenes, las primeras capas ocultas son responsables de las características de nivel superior, como bordes, formas o límites. Se encarga de combinar todas las entradas en un solo valor de salida, sobre el cual se aplicará la función de activación. Generalmente, esto se realiza mediante una suma ponderada de las entradas, es decir, multiplicando cada entrada por su peso correspondiente y sumando los resultados.

- **Capa de salida:** la capa de salida toma como entrada las activaciones generadas por las capas ocultas previas y produce la **predicción final** del modelo, ya sea una clase en un problema de clasificación o un valor continuo en un problema de regresión. Esta salida de la neurona i (out) se corresponde con el valor resultante de la **función de salida**. Si la función de activación está por debajo de un umbral determinado, ninguna salida se pasa a la neurona subsiguiente. Normalmente, no

cualquier valor es permitido como salida de una neurona, por lo tanto, los valores de salida suelen estar comprendidos en rangos como $[0, 1]$ o $[-1, 1]$. También pueden ser binarios $\{0, 1\}$ o $\{-1, 1\}$.

Dos de las funciones de salida más comunes son:

- **Identidad:** la salida es la misma que la entrada. Es la función más sencilla.
- **Binaria:**

$$\text{out}_i = \begin{cases} 1 & \text{si } \text{act}_i \geq \xi_i \\ 0 & \text{de lo contrario} \end{cases}$$

donde ξ_i es el umbral.

En general, para problemas de clasificación o regresión, la capa de salida suele contener una sola neurona, aunque esto puede variar dependiendo del tipo de problema. Por ejemplo:

- En clasificación multiclase, se usan tantas neuronas de salida como clases haya, aplicando una función *softmax*.
- En regresión, normalmente se emplea una única neurona con función de activación lineal.

La estructura completa de una red neuronal y las capas que la conforman se puede ver en la imagen [2.1](#).

Proceso de Aprendizaje

Las redes neuronales tienen la capacidad de aprender: cambiando la distribución de pesos es posible aproximar una función representativa de los patrones en la entrada. El aprendizaje en las redes neuronales depende en gran medida del número de capas ocultas, la cantidad de nodos en cada una de ellas, la dirección del flujo de señales (hacia delante o recurrente) y la inicialización de los pesos de conexión.

Una de las arquitecturas más utilizadas es la **red neuronal multicapa de avance directo** (multi-layer feedforward network) [\[1\]](#). Para entrenar este tipo de redes, se emplea comúnmente el **algoritmo de retropropagación del error (backpropagation)** [\[2\]](#), que permite ajustar los pesos internos del modelo minimizando el error entre la salida predicha y la salida deseada, mediante la aplicación del método del descenso del gradiente.

Algoritmo de Retropropagación del Error

El algoritmo de retropropagación se basa en el **descenso del gradiente** [\[3\]](#) para ajustar los pesos de una red neuronal multicapa. Su objetivo es minimizar una función de error

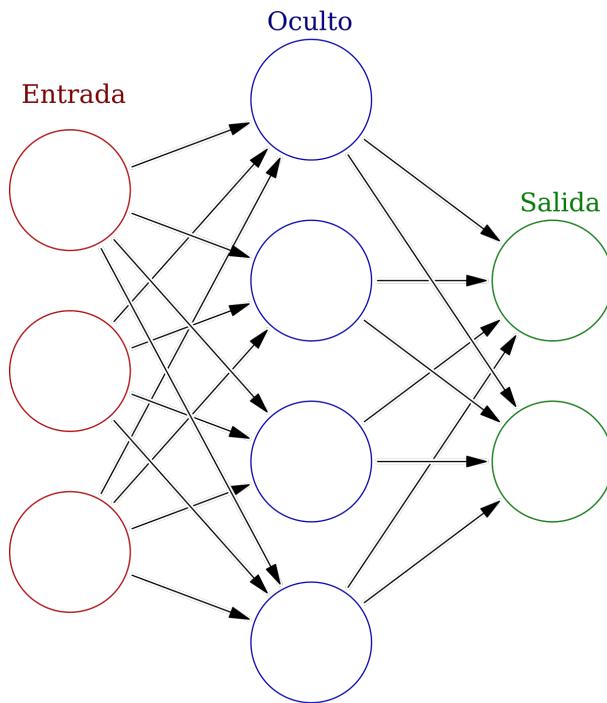


Figura 2.1. Estructura básica de una Red Neuronal Artificial. Fuente: [Wikipedia](#)

E , típicamente definida como el error cuadrático medio entre las salidas deseadas t_k y las salidas reales de la red $z_{\text{out},k}$.

Cálculo del Gradiente en la Capa de Salida

Para ajustar los pesos que conectan una neurona j de la capa oculta con una neurona k de la capa de salida, se calcula la derivada parcial de la función de error E con respecto al peso w'_{jk} :

$$\frac{\partial E}{\partial w'_{jk}} = \frac{\partial}{\partial w'_{jk}} \left\{ \frac{1}{2} \sum_{k=1}^n (t_k - f_z(z_{\text{in},k}))^2 \right\}$$

donde:

- t_k : salida deseada de la neurona de salida k ,
- f_z : función de activación en la capa de salida,
- $z_{\text{in},k}$: entrada neta a la neurona k en la salida,

- w'_{jk} : peso que conecta la neurona j (oculta) con la k (salida).

Aplicando la regla de la cadena:

$$\frac{\partial E}{\partial w'_{jk}} = -(t_k - z_{\text{out},k}) \cdot f'_z(z_{\text{in},k}) \cdot \frac{\partial}{\partial w'_{jk}} \left(\sum_{i=0}^n y_{\text{out},i} \cdot w_{ik} \right)$$

Dado que la entrada neta $z_{\text{in},k}$ depende linealmente de w'_{jk} , se tiene:

$$\frac{\partial z_{\text{in},k}}{\partial w'_{jk}} = y_{\text{out},j}$$

por tanto, el gradiente queda expresado como:

$$\frac{\partial E}{\partial w'_{jk}} = -(t_k - z_{\text{out},k}) \cdot f'_z(z_{\text{in},k}) \cdot y_{\text{out},j}$$

Definición del Término de Error δ'_k

Definimos el **término de error local** para la neurona de salida k como:

$$\delta'_k = -(t_k - z_{\text{out},k}) \cdot f'_z(z_{\text{in},k})$$

Esto permite reescribir el gradiente como:

$$\frac{\partial E}{\partial w'_{jk}} = \delta'_k \cdot y_{\text{out},j}$$

Reglas de Actualización para Pesos y Bias

Siguiendo el descenso del gradiente, los pesos y bias se actualizan de la siguiente forma:

- **Para los pesos:**

$$\begin{aligned} \Delta w'_{jk} &= -\alpha \cdot \frac{\partial E}{\partial w'_{jk}} = -\alpha \cdot \delta'_k \cdot y_{\text{out},j} \\ w'_{jk}^{(\text{new})} &= w'_{jk}^{(\text{old})} + \Delta w'_{jk} \end{aligned}$$

- **Para el bias:**

$$\Delta w'_{0k} = -\alpha \cdot \delta'_k$$

$$w'_{0k}^{(\text{new})} = w'_{0k}^{(\text{old})} + \Delta w'_{0k}$$

donde:

- α es la tasa de aprendizaje,
- $y_{\text{out},j}$ es la salida de la neurona j de la capa oculta,
- δ'_k representa el error en la neurona de salida k .

2.1.1. Convolutional Neural Network

Las [Convolutional Neural Network \(CNN\)](#) [4] tienen el mismo objetivo que las redes neuronales artificiales: aprender representaciones útiles a partir de los datos. Sin embargo, la principal diferencia radica en que están diseñadas específicamente para recibir como input datos que presentan secuencialidad, como imágenes o señales de audio.

En el caso de las imágenes, las [Convolutional Neural Network \(CNN\)](#) aprovechan la disposición bidimensional de los píxeles para detectar patrones locales como bordes, texturas o formas. Esto nos permite codificar características específicas de la imagen en la arquitectura, lo que hace que la red sea más fácil de usar y requiera menos parámetros entrenables.

Arquitectura de las CNN

Una [CNN](#) se construye como una secuencia de capas (Figura 2.2, donde cada capa transforma un volumen de activaciones en otro a través de una función diferenciable. Las CNNs están compuestas principalmente por tres tipos de capas: capas convolucionales, capas de pooling y capas completamente conectadas. Al apilar estas capas de manera secuencial, se conforma la arquitectura completa de la red convolucional, permitiendo extraer representaciones jerárquicas y cada vez más abstractas de los datos de entrada.

Capa Convolucional

Las capas convolucionales son el núcleo de estas redes. Su función es aplicar **filtros** (también llamados *kernels*) sobre regiones locales del input para detectar patrones espaciales. Estos filtros son pequeños (por ejemplo, 3×3 o 5×5), pero se extienden a lo largo de toda la profundidad del input.

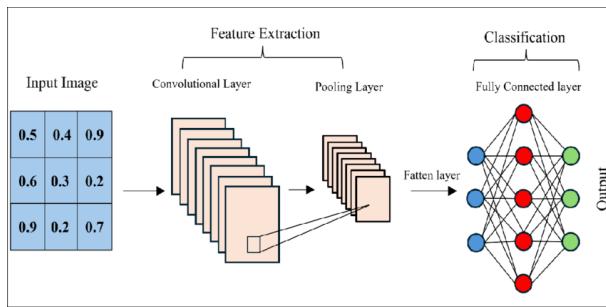


Figura 2.2. Capas de una CNN. Fuente: [ResearchGate](#)

Durante la convolución (Figura 2.3), el filtro se desplaza por la imagen de entrada (controlado por el *stride*), realizando una operación de producto punto entre los valores del filtro y los de la región actual del input, generando un mapa de activación que se puede visualizar. A este resultado se le suele aplicar una función de activación no lineal, como ReLU.

- **Profundidad:** Número de filtros aplicados (cada uno aprende a detectar una característica distinta).
- **Stride:** Cantidad de píxeles que se desplaza el filtro en cada paso.
- **Padding:** Número de ceros añadidos en los bordes para mantener la dimensión del output.

Cada filtro genera un **mapa de activación**, que representa las regiones del input donde se detecta la característica aprendida por ese núcleo. Todos los mapas generados por los distintos filtros se **apilan a lo largo de la dimensión de profundidad**, formando así el volumen de salida completo de la capa convolucional. Este volumen conserva las relaciones espaciales del input original, pero codifica distintas representaciones de alto nivel, útiles para tareas posteriores como la clasificación.

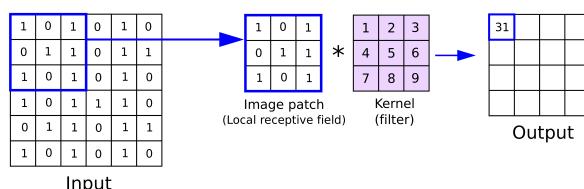


Figura 2.3. Proceso de convolución. Fuente: [Medium](#)

Capa de Pooling

La capa de *pooling* tiene como objetivo reducir las dimensiones espaciales (ancho y alto)

de las activaciones, disminuyendo así la cantidad de parámetros y el coste computacional.

El tipo más común es el **max pooling** (Figura 2.4, que selecciona el valor máximo dentro de una ventana (típicamente de tamaño 2×2) que se desliza por el mapa de activación con un determinado *stride* (normalmente 2)).

- Reduce el riesgo de *overfitting*.
- Mejora la *invarianza traslacional* del modelo.



Figura 2.4. Max-pooling. Fuente: [Medium](#)

Capas Completamente Conectadas

Estas capas se colocan normalmente al final de la red. Cada neurona de una capa completamente conectada está conectada a todas las neuronas de la capa anterior. Su función es integrar la información extraída por las capas anteriores y generar las salidas finales, como las probabilidades de clasificación (Figura 2.5).

También se puede aplicar activación ReLU en estas capas, y la capa final suele utilizar *softmax* en tareas de clasificación multiclase.

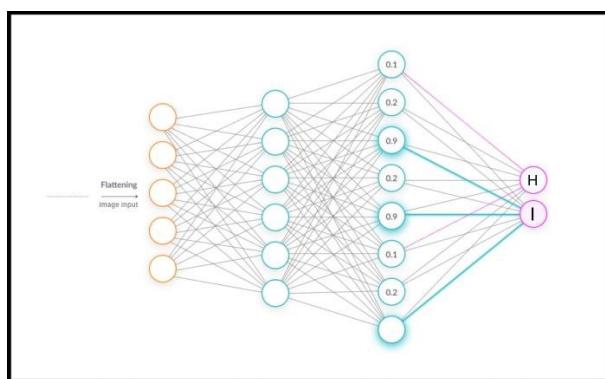


Figura 2.5. Capa completamente conectada. Fuente: [Medium](#)

Arquitectura típica

Una CNN típica organiza estas capas de la siguiente manera:

$$\begin{aligned} [\text{Convolución} + \text{ReLU}] &\rightarrow [\text{Pooling}] \rightarrow \dots \text{ (repetido varias veces)} \\ &\rightarrow [\text{FC} + \text{ReLU}] \rightarrow [\text{Output}] \end{aligned}$$

Este diseño jerárquico permite que las capas iniciales aprendan características locales simples (como bordes), mientras que las capas profundas aprenden representaciones más abstractas (formas, objetos, etc.).

2.2. Modelos generativos

El objetivo del aprendizaje supervisado es relativamente sencillo, asociar nuevos inputs con sus correspondientes outputs. Por ejemplo, la imagen de un perro es clasificada en la categoría PERRO o similar.

Por el contrario, el aprendizaje no supervisado presenta un objetivo más difuso. En términos generales, el objetivo del aprendizaje no supervisado es aprender algo útil examinando un conjunto de datos que contiene ejemplos de entrada no etiquetados.

Una de las aproximaciones más relevantes dentro del aprendizaje no supervisado es el **modelado generativo**. En este enfoque, los ejemplos de entrenamiento x se asumen provenientes de una distribución desconocida $p_{\text{data}}(x)$, y el objetivo del algoritmo es aprender una distribución modelo $p_{\text{model}}(x)$ que aproxime lo mejor posible a la distribución real de los datos.

Una forma directa de abordar este problema consiste en definir una función $p_{\text{model}}(x; \theta)$, controlada por un conjunto de parámetros θ , y buscar los valores de estos parámetros que minimicen la diferencia entre p_{data} y p_{model} . Una estrategia común para lograr esto es la **estimación por máxima verosimilitud**, que consiste en minimizar la divergencia de **Kullback-Leibler (KL)** entre ambas distribuciones. Un ejemplo clásico de esta técnica es la estimación de la media de una distribución gaussiana a partir del promedio de un conjunto de observaciones.

Aunque el modelado explícito de densidades ha sido efectivo en estadística tradicional, especialmente cuando se emplean formas funcionales simples y se trabaja con pocos atributos, el auge del aprendizaje profundo ha motivado la investigación de modelos

con formas funcionales mucho más complejas. En este contexto, el uso de redes neuronales profundas para generar datos suele dar lugar a funciones de densidad difíciles o incluso imposibles de calcular de forma exacta.

Para hacer frente a esta **intractabilidad**, históricamente se han propuesto dos enfoques principales: (1) diseñar modelos cuya función de densidad sea tractable [5], y (2) desarrollar algoritmos de aprendizaje basados en aproximaciones computacionalmente viables de funciones de densidad intratables [6]. Sin embargo, ambos caminos presentan limitaciones, especialmente en tareas como la generación de imágenes realistas de alta resolución, donde los resultados aún no alcanzan la calidad deseada.

Esta limitación ha motivado la exploración de un tercer enfoque, centrado no en evaluar directamente la probabilidad de una muestra, sino en la capacidad del modelo para generar nuevas muestras a partir de la distribución aprendida p_{model} . Esta propiedad resulta especialmente valiosa en aplicaciones creativas como la generación de imágenes, donde el objetivo principal es la síntesis de contenido novedoso y visualmente coherente.

2.2.1. Generative Adversarial Network

Las **Generative Adversarial Network** son un tipo de modelo generativo introducido por Ian Goodfellow et al. en 2014 [7]. Estas redes se enmarcan dentro del aprendizaje no supervisado y tienen como objetivo aprender la distribución de probabilidad subyacente de un conjunto de datos $p_{\text{data}}(x)$ para poder generar nuevas muestras $x \sim p_{\text{model}}(x)$ que resulten indistinguibles de las reales.

A diferencia de los enfoques clásicos basados en estimación explícita de funciones de densidad como la estimación por máxima verosimilitud o los modelos variacionales, las GANs se basan en un enfoque **implícito**, en el que no se modela directamente la función $p_{\text{model}}(x)$, sino que se aprende a generar muestras mediante un proceso de entrenamiento competitivo entre dos redes neuronales.

Existen muchas variantes como **Deep Convolutional Generative Adversarial Network (DCGAN)** (introduce redes convolucionales profundas en el generador y el discriminador), **CycleGAN** [8] (diseñado para tareas de transferencia de estilo no supervisadas) o **StyleGAN** [9] (incorpora un control detallado sobre características visuales).

Arquitectura y funcionamiento

El modelo **GAN** está compuesto por dos redes principales que se entrenan simultáneamente (Figura 2.6):

- **Generador (G)**: recibe como entrada un vector aleatorio z proveniente de una distribución conocida (generalmente una gaussiana multivariada o uniforme), y lo transforma en una muestra sintética $x = G(z)$, con el objetivo de imitar el comportamiento de las muestras reales.
- **Discriminador (D)**: recibe como entrada una imagen (real o generada) y predice la probabilidad de que provenga del conjunto de datos reales. En términos prácticos, actúa como un clasificador binario.

Tanto el generador como el discriminador son redes neuronales. La salida del generador se conecta directamente a la entrada del discriminador. A través de la retropropagación, la clasificación del discriminador proporciona un indicador que el generador usa para actualizar sus pesos.

Ambas redes se entrenan en un esquema de juego competitivo, inspirado en la teoría de juegos: el generador intenta engañar al discriminador generando imágenes realistas, mientras que el discriminador intenta mejorar su capacidad de distinguir entre imágenes reales y falsas.

Una analogía común para entender GANs es la relación entre **falsificadores** y **detectives**:

- El generador actúa como un falsificador que intenta crear billetes indistinguibles de los reales.
- El discriminador actúa como un agente que trata de detectar las falsificaciones.

Con el tiempo, si ambos mejoran simultáneamente, se alcanza un equilibrio de Nash en el que el generador produce muestras tan realistas que el discriminador ya no puede distinguirlas de las verdaderas ($p_{\text{model}} \approx p_{\text{data}}$).

Entrenamiento del Generador (G)

El entrenamiento del generador implica una integración más estrecha con el discriminador. Como el generador no tiene acceso directo a una verdad objetiva, su objetivo es engañar al discriminador. Para ello:

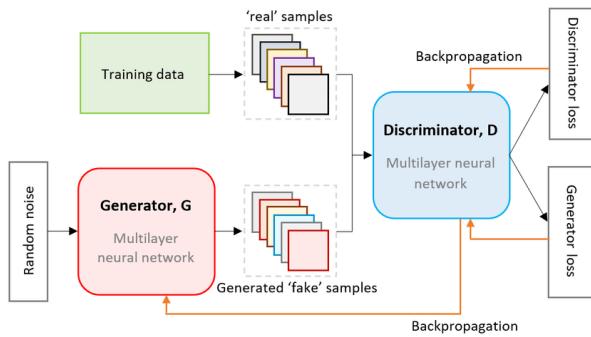


Figura 2.6. Ejemplo de la arquitectura de una GAN. Fuente: [ResearchGate](#)

1. Se toma una muestra de ruido aleatorio $z \sim p(z)$, comúnmente de una distribución uniforme o normal de baja dimensión.
2. El generador transforma este ruido en una instancia sintética $G(z)$.
3. Esta instancia se pasa al discriminador, que emite una clasificación (real o falsa).
4. Se realiza la retropropagación desde la salida del discriminador, a través de toda la red, hasta los parámetros del generador. El discriminador permanece congelado durante esta etapa para evitar que el generador aprenda sobre una función objetivo cambiante.
5. Se actualizan únicamente los pesos del generador.

Entrenamiento del Discriminador (D)

Durante la fase de entrenamiento del discriminador, se le presentan instancias tanto reales (extraídas del conjunto de datos) como falsas (producidas por el generador). El discriminador calcula una función de pérdida que penaliza las clasificaciones erróneas, es decir, clasificar datos reales como falsos o viceversa. A partir de esta pérdida, el discriminador actualiza sus pesos mediante retropropagación, sin modificar los parámetros del generador en este paso.

Formulación matemática

La dinámica de entrenamiento de una GAN puede entenderse como un juego de suma cero entre dos agentes. Formalmente, se define una función de valor $V(G, D)$ tal que:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log D(x)] + \mathbb{E}_{z \sim p(z)} [\log(1 - D(G(z)))]$$

donde:

- $D(x)$ es la probabilidad asignada por el discriminador a que x sea real.
- $G(z)$ es una muestra generada a partir del ruido z .
- $p(z)$ es la distribución previa del espacio latente (por ejemplo, $\mathcal{N}(0, I)$).

Durante el entrenamiento:

- El discriminador D busca maximizar la función de valor, clasificando correctamente reales y falsos.
- El generador G busca minimizarla, generando muestras que el discriminador clasifique como reales.

Funciones de pérdida

El entrenamiento de **GAN** se realiza mediante *descenso por gradiente alternado*: en cada iteración se actualizan los parámetros de D y G por separado, usando funciones de pérdida derivadas de la formulación anterior. Existen dos variantes comunes para la pérdida del generador:

- **Minimax GAN:** $J_G = -\mathbb{E}_{z \sim p(z)}[\log(1 - D(G(z)))]$ [7]
- **Pérdida de Wasserstein:** Es la función de pérdida predeterminada para los estimadores de TF-GAN

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim p_{\text{data}}}[D(x)] - \mathbb{E}_{z \sim p(z)}[D(G(z))]$$

[10]

donde \mathcal{D} representa el conjunto de funciones 1-Lipschitz.

2.2.2. Deep Conditional Generative Adversarial Networks

Como hemos visto, las **Generative Adversarial Network (GAN)** han demostrado ser altamente eficaces en la generación de imágenes realistas. Sin embargo, uno de los principales desafíos que presentan es la **inestabilidad durante el entrenamiento**, lo que puede provocar que el generador produzca imágenes incoherentes o sin valor visual.

Para mitigar estas limitaciones, se propuso una variante conocida como **Deep Convolutional Generative Adversarial Network (DCGAN)** [11], que combina la arquitectura básica de una **GAN** con **CNN**. Esta combinación permite aprovechar la capacidad de las **Convolutional Neural Network** para extraer y procesar características espaciales relevantes en las imágenes, al tiempo que se mantiene la estructura de entrenamiento competitivo entre generador y discriminador.

De esta forma, se logra una mejora significativa en la estabilidad del entrenamiento y en la calidad de las imágenes generadas. En particular:

- El **generador** se transforma en una red deconvolucional profunda, que toma como entrada un vector latente $z \in \mathbb{R}^d$ y lo transforma, mediante una serie de capas convolucionales transpuestas, en una imagen sintética.
- El **discriminador** adopta una estructura de red convolucional clásica, que permite identificar si una imagen proviene del conjunto real o ha sido generada artificialmente.

Uno de los beneficios más relevantes de esta arquitectura es que hace posible **visualizar los filtros aprendidos** por la red durante el entrenamiento. Se ha observado empíricamente que ciertos filtros del generador aprenden a representar rasgos visuales específicos, como bordes, texturas u objetos concretos. Esto sugiere que el modelo está desarrollando una forma jerárquica de representación, en la que cada capa va componiendo estructuras más complejas a partir de elementos visuales más simples.

En resumen, la idea principal del **DCGAN** respecto a la **GAN** original radica en el uso de **capas convolucionales y convolucionales transpuestas** en lugar de capas completamente conectadas.

Arquitectura de una DCGAN

La arquitectura de las **DCGAN** incorpora una serie de principios que han demostrado mejorar la estabilidad y la calidad del entrenamiento en redes generativas profundas. Estos principios, basados en investigaciones previas en arquitecturas convolucionales (como las Fully Convolutional Networks), se resumen en los siguientes pilares fundamentales:

- Utilizar una **red completamente convolucional** [12] tanto en el generador como en el discriminador, sin capas de *pooling* ni capas completamente conectadas.

- En lugar de *pooling*, usar únicamente **convoluciones con stride** para reducir el tamaño espacial. Por ejemplo, si se desea que la salida tenga la mitad del tamaño que la entrada, se puede aplicar una convolución con `stride=2`.
- Añadir capas de **normalización por lotes (Batch Normalization)** para mejorar la estabilidad del entrenamiento. [13]
- **Eliminar las capas ocultas completamente conectadas** en ambas redes.
- Utilizar la función de activación **ReLU** [14] en el generador para todas las capas, excepto en la capa de salida, donde se emplea **Tanh**.
- Utilizar la función de activación **LeakyReLU** [15], [16] en el discriminador para todas las capas.

Generador

El generador de una de este tipo de arquitecturas típica recibe como entrada un vector de ruido aleatorio $z \in \mathbb{R}^{100}$, muestreado de una distribución uniforme o normal. A partir de este vector latente, el generador aplica una secuencia de cuatro capas convolucionales transpuestas (también conocidas como convoluciones con paso fraccionario) con el objetivo de **sobremuestrear** progresivamente la entrada y generar una imagen sintética (Figura 2.7).

Durante este proceso, las dimensiones espaciales de los mapas de características aumentan (por ejemplo, de 4×4 a 8×8 , 16×16 , etc.), mientras que la profundidad (número de canales) se reduce gradualmente. Por ejemplo, tras la primera operación de convolución, el número de canales de salida puede pasar de 512 a 256, dependiendo de la configuración exacta del modelo.

Todas las capas convolucionales transpuestas utilizan un *stride* de 2, lo que permite duplicar las dimensiones espaciales en cada paso. Además, se aplica **normalización por lotes (Batch Normalization)** tras cada capa (excepto en la salida) para estabilizar el proceso de entrenamiento, mejorar el flujo del gradiente y mitigar problemas derivados de una mala inicialización.

Como función de activación, se emplea ReLU en todas las capas del generador, excepto en la capa de salida, donde se utiliza la función tangente hiperbólica (`tanh`). Esta última activa los valores de píxel dentro del rango $[-1, 1]$, lo que resulta coherente con las imágenes reales preprocesadas con normalización estándar.

La salida final del generador tiene típicamente dimensiones de $64 \times 64 \times 3$, correspondiente a una imagen RGB generada a partir de la distribución aprendida.

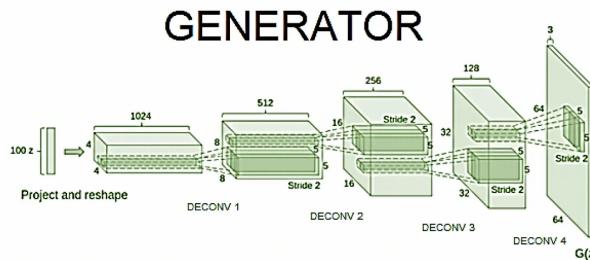


Figura 2.7. Arquitectura del Generador de una DCGAN. Fuente: [Medium](#)

Discriminador

El discriminador recibe como entrada una imagen, real o generada por el generador, y tiene como objetivo determinar si esta proviene del conjunto de datos reales o ha sido sintetizada artificialmente. Su arquitectura consiste en una secuencia de **capas convolucionales con stride**, que realizan un **submuestreo progresivo** de las dimensiones espaciales de la imagen mientras aumentan la profundidad de los mapas de características (Figura 2.8).

Esta reducción espacial permite extraer representaciones jerárquicas de alto nivel que facilitan la tarea de clasificación binaria. A lo largo de las capas del discriminador, se aplica **normalización por lotes** en todas las capas, con excepción de la capa de entrada. Esta técnica mejora la estabilidad del entrenamiento y acelera la convergencia al reducir la covarianza interna del modelo.

A continuación de cada normalización por lotes, se emplea una función de activación Leaky ReLU, que introduce una pequeña pendiente para los valores negativos. A diferencia de la ReLU tradicional, que anula completamente las entradas negativas, la Leaky ReLU permite mantener un pequeño flujo de gradiente incluso en esa región, lo cual ayuda a prevenir el problema de la muerte de neuronas y mejora el aprendizaje en presencia de ruido o valores atípicos.

Finalmente, la última capa del discriminador es una única neurona con función de activación sigmoidea, que produce una probabilidad en el rango $[0, 1]$. Este valor representa la confianza del modelo en que la imagen de entrada sea real (cercana a 1) o generada (cercana a 0).

La arquitectura completa de una red **DCGAN** es la que se muestra en la imagen 2.9.

Entrenamiento

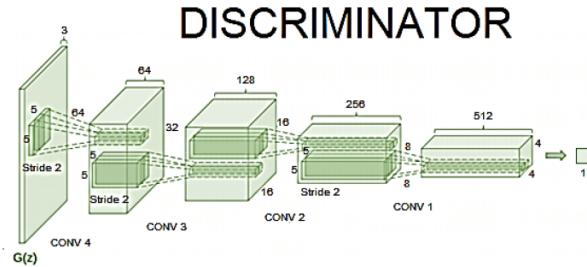


Figura 2.8. Arquitectura del Discriminador de una DCGAN. Fuente: [Medium](#)

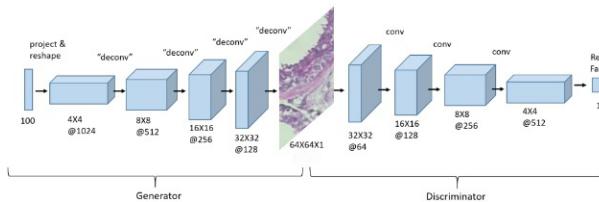


Figura 2.9. Arquitectura de una red DCGAN. Fuente: [UPM](#)

Durante el entrenamiento, el generador recibe retroalimentación del discriminador a medida que ambos modelos se optimizan de forma conjunta mediante un esquema de entrenamiento adversario. El objetivo principal es que el generador aprenda a producir imágenes cada vez más realistas, hasta el punto de resultar indistinguibles de las imágenes del conjunto de entrenamiento.

Ambos modelos utilizan como función de pérdida la **entropía cruzada binaria** presentado por primera vez en el paper de Shannon [17], comúnmente empleada en tareas de clasificación binaria. Esta función cuantifica la discrepancia entre las predicciones del modelo y las etiquetas reales, penalizando con mayor intensidad aquellas predicciones incorrectas con alta confianza. Matemáticamente, se define como:

$$\mathcal{L}_{\text{BCE}}(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

donde:

- $y_i \in \{0, 1\}$ es la etiqueta verdadera del dato i ,
- $\hat{y}_i \in (0, 1)$ es la probabilidad predicha por el modelo,
- N es el número total de muestras.

Para minimizar esta función de pérdida, ambos modelos utilizan el **optimizador Adam**, una técnica que combina las ventajas de dos métodos previos: RMSprop y el descenso de gradiente estocástico (SGD) con momento. Adam adapta la tasa de aprendizaje para cada parámetro de forma individual en función de momentos de primer y segundo orden del gradiente.

El procedimiento de actualización de parámetros en Adam está definido por:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1 - \beta_1) \nabla \mathcal{L}_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2) (\nabla \mathcal{L}_t)^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\ \theta_{t+1} &= \theta_t - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned}$$

[18]

donde:

- θ_t representa los parámetros del modelo en el paso t ,
- α es la tasa de aprendizaje,
- β_1, β_2 son coeficientes de decaimiento de momento típicamente fijados en 0,9 y 0,999,
- ϵ es un pequeño número para evitar división por cero,
- $\nabla \mathcal{L}_t$ es el gradiente de la pérdida en el paso t .

A medida que el entrenamiento avanza, el generador se ajusta para minimizar la pérdida derivada de las predicciones del discriminador, refinando así su capacidad de generar imágenes plausibles. El objetivo final es alcanzar un equilibrio en el que el discriminador ya no pueda discernir con precisión si una imagen proviene del conjunto real o ha sido generada sintéticamente, es decir, cuando $D(G(z)) \approx 0,5$.

2.2.3. Autoencoder

Un **autoencoder** es un tipo específico de **red neuronal**, que está diseñado principalmente para **codificar la entrada** en una representación comprimida y significativa, y luego **decodificarla** de manera tal que la **entrada reconstruida** sea lo más similar posible a la original [19].

En el artículo [20] se presenta la problemática que consiste en aprender las funciones:

$$A : \mathbb{R}^n \rightarrow \mathbb{R}^p \quad (\text{codificador}) \quad \text{y} \quad B : \mathbb{R}^p \rightarrow \mathbb{R}^n \quad (\text{decodificador})$$

que satisfacen

$$\arg \min_{A,B} \mathbb{E} [\Delta(\mathbf{x}, B \circ A(\mathbf{x}))], \quad (1)$$

donde \mathbb{E} es la esperanza sobre la distribución de \mathbf{x} , y Δ es la función de pérdida de reconstrucción, que mide la distancia entre la salida del decodificador y la entrada original. Usualmente, esta distancia se establece como la norma ℓ_2 .

Arquitectura de un Autoencoder

Los **autoencoders (AE)** se componen de dos elementos (Figura 2.10):

- **Codificador:** se encarga de comprimir los datos de entrada eliminando el ruido y extrayendo características relevantes.
- **Decodificador:** toma como entrada únicamente la representación comprimida del codificador y busca reconstruir los datos originales con la mayor fidelidad posible. En general, la arquitectura del decodificador es una imagen especular del codificador.

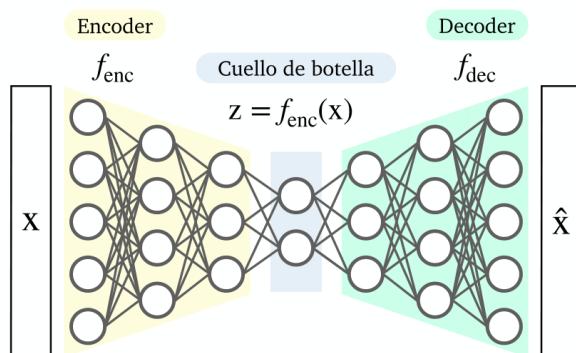


Figura 2.10. Arquitectura de un autoencoder. Fuente: [ResearchGate](#)

Funcionamiento

Cada componente del [autoencoder](#) está formado por capas totalmente conectadas. La capa de entrada recoge los valores de cada muestra (por ejemplo, los píxeles de una imagen). Una o varias capas intermedias que transforman los datos originales en un espacio latente de menor dimensión, también llamado cuello de botella (*bottleneck*). La cantidad de neuronas en esta capa intermedia es significativamente menor que en la capa de entrada, lo cual permite realizar una compresión de la información. Esta dimensión reducida se puede ajustar como un hiperparámetro, determinando el grado de compresión y la pérdida aceptada en la reconstrucción. Finalmente, la capa de salida reconstruye los datos, teniendo la misma dimensión que la capa de entrada. La calidad de la reconstrucción se mide a través del error de reconstrucción, que se calcula mediante una función de pérdida, típicamente la [Mean Squared Error \(MSE\)](#) (error cuadrático medio), entre los valores originales de entrada y los valores generados en la salida.

El [autoencoder](#) puede verse como una generalización no lineal del *pca*. Mientras que [Análisis de Componentes Principales \(PCA\)](#) busca encontrar un hiperplano de baja dimensión donde residen los datos, el autoencoder es capaz de aprender una variedad no lineal.

2.2.4. Variational Autoencoders (VAE)

Los [autoencoders](#) son muy eficientes a la hora de aprender representaciones comprimidas de los datos, permitiendo tareas como la reducción de dimensionalidad o la eliminación de ruido. Sin embargo, presentan limitaciones a la hora de generar nuevas muestras, ya que el espacio latente no está estructurado probabilísticamente. Esta deficiencia impide un muestreo coherente y controlado, dificultando tareas como la generación de datos o la interpolación en el espacio latente.

Los [Variational Autoencoders](#) presentados en [6] abordan esta limitación mediante la generación de datos a partir de una distribución probabilística. Esto permite a un [VAE](#) no solo reconstruir con precisión la entrada original exacta, sino también utilizar la inferencia variacional para generar nuevas muestras de datos que se parezcan a los datos de entrada originales. Esta diferencia se puede apreciar en la figura 2.11.

En vez de codificar las variables latentes de los datos como un valor discreto fijo, se codifican como una distribución de probabilidad $p(z)$, también conocido como *distribución a priori*. Durante la inferencia variacional, es decir, el proceso generativo de nuevos

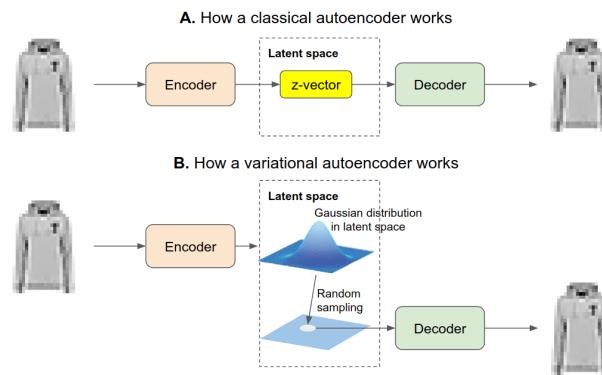


Figura 2.11. Comparación entre los autoencoders tradicionales con los autoencoders variacionales. Fuente: [Manning](#)

datos, se utiliza esta distribución para calcular la *distribución posterior* $p(x|z)$, es decir el valor de las variables observables x dado un valor para la variable latente z .

Para cada atributo latente de los datos de entrenamiento, los **VAE** codifican dos vectores latentes diferentes: un vector de medias, μ , y un vector de desviaciones típicas, σ . En esencia, estos dos vectores representan el rango de posibilidades de cada variable latente y la varianza esperada dentro de cada rango de posibilidades.

Como cualquier otro **autoencoder**, los **VAE** emplean el **error de reconstrucción**. Sin embargo, para la inferencia variacional esta regularización puede resultar demasiado estricta. Por lo tanto, estos modelos incorporan otro término de regularización: la divergencia de **KL**.

Una vez entrenado, el decodificador de un **VAE** es capaz de generar imágenes tomando muestras directamente del espacio latente. Si se toman puntos que coinciden exactamente con las codificaciones latentes de las imágenes del conjunto de entrenamiento, el resultado simplemente replica esas imágenes. Sin embargo, el verdadero interés está en generar nuevas imágenes a partir de puntos intermedios o completamente nuevos en el espacio latente.

Para que esto sea posible, el espacio latente debe estar estructurado de forma que cualquier punto que se muestree en él sea útil y coherente. Esto implica que debe presentar dos propiedades clave:

- **Continuidad:** puntos cercanos entre sí en el espacio latente deben generar salidas similares.
- **Integridad:** cualquier punto del espacio latente debe producir una salida válida

y coherente al ser decodificado.

El problema es que si solo se entrena el modelo minimizando el error de reconstrucción, no se fuerza al espacio latente a cumplir ninguna de estas dos propiedades. El modelo aprenderá a codificar bien las imágenes originales, pero no tendrá ninguna restricción sobre cómo se organizan las codificaciones en ese espacio.

Aquí es donde entra en juego el segundo término de regularización: la **divergencia de KL**. Esta divergencia mide cuánto difiere la distribución latente aprendida por el encoder, $q_\phi(z|x)$, de una distribución normal estándar $p(z) = \mathcal{N}(0, I)$. Minimizar esta divergencia fuerza al modelo a que todas las codificaciones latentes estén distribuidas de forma continua y cercana a una gaussiana, lo que permite interpolar entre ellas y muestrear puntos nuevos con sentido.

La divergencia **KL** presenta un problema: produce ecuaciones intractables, es decir, que su resolución conlleva mucho tiempo computacional o incluso no se puede realizar de forma exacta debido a la necesidad de integrar sobre distribuciones complejas o de alta dimensión. Por ello, se introdujo **Evidence Lower Bound (ELBO)**, una técnica que permite aproximar la verosimilitud marginal del modelo desde abajo. Esta técnica transforma aquellas operaciones intractables en problemas de optimización, haciendo posible trabajar con funciones objetivo que sí se pueden calcular. Así, la inferencia bayesiana se convierte en una tarea de optimización, y se pueden resolver con ayuda de otros métodos como el descenso de gradiente.

A pesar de que **ELBO** convierte el problema de inferencia en uno de optimización, todavía persiste una dificultad técnica: no es posible derivar directamente a través del muestreo de la variable latente z , ya que este proceso no es diferenciable. Para resolver este inconveniente, los autores de Kingma y Welling [6] introducen una técnica conocida como **truco de reparametrización**, que permite reformular el muestreo de z como una operación determinista y diferenciable con respecto a los parámetros del modelo.

En lugar de muestrear directamente z desde una distribución como $\mathcal{N}(\mu, \sigma)$, se utiliza una variable de ruido auxiliar $\epsilon \sim \mathcal{N}(0, 1)$, y se define:

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

De este modo, el muestreo se convierte en una función determinista y diferenciable respecto a los parámetros del modelo. Esto permite calcular la esperanza y su gradiente

de forma directa, lo que hace posible el uso de backpropagation.

Además, se puede reformular la [ELBO](#) y calcularla con Monte Carlo usando simplemente una única muestra $\epsilon \sim \mathcal{N}(0, 1)$.

$$\log p(x|z) + \log p(z) - \log q(z|x)$$

donde z es muestrada de $q(z|x)$.

2.2.5. Modelos de Difusión

Los modelos de difusión son modelos generativos probabilísticos cuya idea principal se basa en difuminar progresivamente una imagen mediante ruido aleatorio y aprender a revertir el proceso de difusión con el objetivo de generar nuevas imágenes (Figura 2.12) han sido implementados con gran éxito en modelos como [Denoising Diffusion Probabilistic Models](#) [21] o [Stable Diffusion](#) [22]. Este concepto es similar al de los [Variational Autoencoders](#) en la manera de optimizar una función objetivo aprendiendo a proyectar la imagen de entrada en un espacio latente y, después, recuperar la imagen original. La diferencia con respecto a los [VAE](#) es que, en vez de, aprender una distribución de los datos, aprende a modelar una serie de distribuciones de ruido en una *cadena de Markov*.

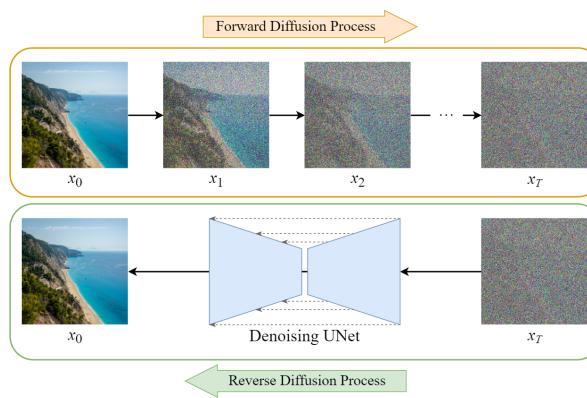


Figura 2.12. Funcionamiento de un modelo de difusión. Fuente: [CSDN](#)

Funcionamiento

La idea de eliminación de ruido fue introducida por primera vez en [23]. Este proceso consiste en dos etapas: la difusión de las imágenes y la reconstrucción.

Proceso de difusión

El objetivo del proceso de difusión es transformar una imagen de datos reales $\mathbf{x}_0 \sim q(\mathbf{x}_0)$

en una variable latente que se asemeje a ruido gaussiano estándar. Partiendo de los datos originales, se añade ruido gaussiano mediante un proceso de Markov progresivamente a lo largo de T pasos:

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I}) \quad (2.1)$$

La distribución conjunta del proceso se factoriza como:

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_{t=1}^T q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \quad (2.2)$$

Aquí $\beta_t \in (0, 1)$ es un hiperparámetro predefinido que controla la cantidad de ruido introducido en cada paso; puede mantenerse constante o cambiar dinámicamente. No obstante, si fijamos dicho valor como $\beta_t = 1$ se puede obtener el valor de \mathbf{x}_t sin necesidad de pasar por todos los pasos intermedios. Esta marginalización se expresa como:

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}) \quad (2.3)$$

donde:

$$\alpha_t = 1 - \beta_t \quad (2.4)$$

$$\bar{\alpha}_t = \prod_{s=1}^t \alpha_s \quad (2.5)$$

De acuerdo con la dinámica de gradiente estocástico de Langevin [24] podemos muestrear los nuevos estados del sistema solo mediante el gradiente de la función de densidad en una cadena de Markov. Esto permite generar directamente una muestra \mathbf{x}_t de la siguiente forma:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I}) \quad (2.6)$$

Proceso de eliminación de ruido

La generación de nuevas imágenes se realiza recorriendo la cadena de forma inversa, desde $\mathbf{x}_T \sim \mathcal{N}(0, \mathbf{I})$ hasta obtener \mathbf{x}_0 . A diferencia del proceso anterior, para estimar el

estado anterior a partir del actual requiere el conocimiento de todos los estados previos, lo cual hace este proceso intractable. Por ende, para calcular la probabilidad $q(\mathbf{x}_{t-1} | \mathbf{x}_t)$ de forma exitosa, es necesario entrenar red neuronal basada en unos pesos y el estado actual.

La distribución de transición inversa se puede modelar de la siguiente forma:

$$p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \quad (2.7)$$

donde $\boldsymbol{\mu}_\theta$ y Σ_θ se parametrizan en función de la predicción del ruido [21]. Durante la inferencia, la imagen se genera aplicando iterativamente el siguiente paso de muestreo:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \cdot \boldsymbol{\varepsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}, \quad \mathbf{z} \sim \mathcal{N}(0, \mathbf{I}) \quad (2.8)$$

Pérdida

Durante el entrenamiento se utiliza la [Mean Squared Error](#) o L1 entre el ruido real $\boldsymbol{\varepsilon}$ y la predicción $\boldsymbol{\varepsilon}_\theta$.

Arquitectura

La arquitectura de los modelos de difusión es muy semejante a la de los [Variational Autoencoders](#): una red convolucional compuesta de la capa de entrada, diversas capas ocultas, incluyendo funciones de activación, y la capa de salida. La única diferencia es que esta arquitectura es que la capa de salida produce dos valores correspondientes con la media y la varianza de la distribución.

2.3. Modelos autoregresivos y variantes recientes

Los modelos autorregresivos predicen automáticamente el siguiente componente de una secuencia al medir las entradas anteriores de la secuencia. Estos modelos generan una imagen píxel a píxel (o bloque a bloque), condicionando cada elemento generado al anterior. Uno de los ejemplos más representativos es [DALL·E](#) [25](OpenAI), que combina codificadores visuales con modelos de lenguaje para generar imágenes realistas

a partir de descripciones textuales. Otros modelos como [VQ-GAN](#) [26] y [27] emplean una codificación discreta de imágenes para permitir una generación más eficiente y controlada. Aunque estos enfoques han demostrado resultados visuales notables, especialmente en tareas de generación condicionada por texto, su uso en el ámbito de la moda aún es limitado debido a la necesidad de grandes cantidades de datos y recursos computacionales para lograr resultados de alta calidad.

2.4. Aplicaciones de IA en la industria de la moda

La [inteligencia artificial](#) junto con las redes sociales han redefinido los límites de la moda y la creatividad. Cada vez es más común encontrar estos elementos presentes durante el proceso creativo, tanto en la etapa inspiracional y experimental como en la etapa de manufacturación. Por ejemplo, existen muchas plataformas que se apoyan en sistemas de recomendación para proporcionar ideas de *outfits* basadas en tus gustos personales. Asimismo, existen modelos capaces de generar ideas de prendas digitales basadas en descripciones textuales o proporcionar sugerencias en cuanto a color, textura y ergonomía de las prendas a partir de una idea base. En la etapa de producción también se pueden integrar sistemas de control de calidad asistidos por [IA](#) o predecir fallos de maquinaria mediante mantenimiento predictivo con sensores [Internet of Things](#) e [IA](#). Finalmente, gracias a las técnicas de minería de datos, se pueden llevar a cabo análisis exhaustivos de tendencias y mercado, proporcionando información valiosa y, así, poder sacar el máximo rendimiento a las creaciones.

Esta sinergia entre la tecnología y la moda ha supuesto una nueva era donde los conocimientos basados en datos, el análisis predictivo y las aplicaciones innovadoras de [IA](#) están revolucionando la forma en que se conciben, diseñan y comercializan las prendas.

3.

Metodología

3.1. Descripción General del Enfoque

Este trabajo se centra en el uso de diferentes técnicas de inteligencia artificial generativa en el proceso creativo en el sector de la moda de lujo, específicamente, en la generación de nuevas prendas a partir de un conjunto de datos visuales extraídos manualmente. El proceso de desarrollo se ha dividido en cuatro etapas principales: preparación del dataset, preprocesamiento de los datos, entrenamiento del modelo y evaluación de los resultados.

El pipeline de desarrollo seguido se ha estructurado en cuatro etapas principales:

1. **Construcción del dataset:** al no existir un conjunto de datos público que recogiera imágenes de moda de lujo con la resolución y características deseadas, se desarrolló un proceso personalizado de recopilación utilizando *web scraping*.
2. **Preprocesamiento de datos:** incluyó la conversión de imágenes a un formato y espacio de color estándar, la normalización y el redimensionamiento, así como la implementación de técnicas de *data augmentation* para enriquecer el conjunto y mejorar la robustez del entrenamiento.
3. **Entrenamiento de modelos generativos:** se implementaron y entrenaron tres enfoques representativos del estado del arte: DCGAN, VAE y modelos de difusión, explorando también variaciones arquitectónicas en algunos de ellos.
4. **Evaluación de resultados:** tanto cualitativa (inspección visual) como cuantitativa (análisis de funciones de pérdida y métricas de convergencia), para determinar la calidad, diversidad y coherencia visual de las imágenes generadas.

Cada una de estas etapas se documenta con detalle en las siguientes secciones, incluyendo decisiones de diseño, herramientas utilizadas, y análisis crítico de los resultados obtenidos.

3.2. Dataset

Dado que ningún dataset público existente recogía las características específicas que se querían para este proyecto, es decir, imágenes de prendas pertenecientes exclusivamente al sector de la moda de lujo, con variedad visual y buena resolución, se optó por construir un dataset propio desde cero utilizando técnicas de *web scraping*.

3.2.1. Página web

Primero, se recurrió a la página web de [Vogue Runway](#) una web derivada de la reconocida revista *Vogue* que está enfocada exclusivamente en los desfiles celebrados de cada temporada alrededor del mundo. Desafortunadamente, para acceder a todo el archivo de imágenes se debe pagar una suscripción, es por eso, que solo se pudo obtener un número limitado e insuficiente de imágenes. La siguiente página web de la que se recolectaron fue [Farfetch](#), una web de compra online en el que se venden los productos de las últimas colecciones de un gran número de marcas de lujo.

3.2.2. Formato de las imágenes

Las imágenes obtenidas de *Vogue Runway* muestran las modelos en la pasarela con las prendas de la colección, estas imágenes son más heterogéneas ya que la imagen no muestra simplemente la prenda, sino que está integrada con el entorno proporcionando información adicional sobre el fondo, la modelo, etc. Esta variedad de información dificulta el proceso de entrenamiento ya que se necesitan modelos más eficientes para captar las características de las prendas pero a su vez, generan imágenes más reales, complejas y coherentes con el contexto de una pasarela. A diferencia de las anteriores, las imágenes de *Farfetch* solo muestran el producto sobre un fondo blanco. La resolución de las imágenes de la primera fuente es de 2560×3840 , mientras que la de la segunda es de 1000×1334 . Las imágenes fueron recolectadas entre abril y mayo de 2025, implementando un script en Python basado en la librería `Selenium` para simular la navegación, gestionar la carga dinámica del contenido y automatizar el guardado. El proceso incluyó control del scroll infinito, aceptación automática de cookies y extracción de los enlaces a las imágenes de cada producto individual. Las descargas finales se realizaron con el gestor `aria2c` para optimizar la velocidad y estabilidad. En total se obtuvieron



(a) Formato de las imágenes obtenidas de Vogue Runway



(b) Formato de las imágenes obtenidas de Farfetch

Figura 3.1. Comparación de ambas imágenes

1843 imágenes.

En la figura 3.1 se puede ver la diferencia entre ambas imágenes.

3.2.3. Técnica empleada

La técnica empleada fue el *web scraping*. Esta práctica consiste en la **extracción automática de contenidos** y datos de sitios web mediante **software especializado** [28]. El proceso de extracción se puede resumir en tres fases:

1. Primero, el fragmento de código utilizado para extraer la información (también conocido como “bot de extracción”) envía una solicitud HTTP GET a un sitio web determinado. Esta solicitud puede ser realizada directamente a través de librerías como `requests` o mediante la simulación de un navegador usando herramientas

como *Selenium*.

2. Cuando el sitio web responde, el scraper analiza el contenido HTML de la página para localizar la información de interés. Esto se realiza mediante técnicas de *parsing* [29], buscando patrones específicos en la estructura del documento (como etiquetas, clases o identificadores) con ayuda de herramientas como BeautifulSoup, lxml o selectores CSS.
3. Tras localizar la información deseada, esta se extrae y se convierte al formato necesario para su posterior análisis (por ejemplo: CSV, JSON, imágenes o bases de datos).

Antes de aplicar esta técnica es fundamental asegurarse de que esté implícitamente permitida en el sitio web deseado. Para ello, se debe revisar el archivo robots.txt, el cual especifica qué partes de la página pueden ser exploradas por bots automatizados.

3.2.4. Implementación

Para llevar a cabo la implementación en Python se utilizó la librería Selenium, que permite automatizar la navegación en sitios web. Dado que la lógica implementada fue similar para ambos casos, y con el fin de evitar repeticiones innecesarias, se va a explicar el proceso general seguido ejemplificándolo con el segundo caso. El script desarrollado realiza una serie de tareas que detallaremos a continuación:

1. **Acceso a la página de categoría:** Se inicializa un navegador Chrome controlado por el script y se accede a la URL correspondiente a la categoría seleccionada dentro del sitio web, Farfetch para ropa de mujer. Esta URL presenta un catálogo de productos que se va cargando dinámicamente conforme el usuario hace scroll.
2. **Gestión de cookies:** Al cargar la página, el sitio muestra un banner de consentimiento de cookies. El script localiza automáticamente el botón de aceptación a través de su selector CSS y simula un clic para aceptar las cookies y poder continuar navegando sin restricciones.
3. **Scroll automático y carga dinámica del contenido:** Dado que el catálogo utiliza scroll infinito, el script implementa una función que simula el desplazamiento vertical progresivo de la página (`window.scrollTo(...)`) en intervalos definidos. Este proceso fuerza la carga de nuevos productos conforme se va alcanzando

el final visible del sitio. Se hace una pausa breve (`time.sleep()`) tras cada desplazamiento para permitir que los nuevos elementos se carguen completamente antes de continuar.

4. **Recolección de enlaces de productos:** Tras cada iteración de scroll, el script detecta todos los elementos HTML que contienen enlaces a productos (`<a>` con clase específica o `data-testid`) y extrae sus URLs. Estos enlaces se almacenan en una lista, evitando duplicados mediante una estructura tipo set.
5. **Acceso y descarga de imágenes de producto:** Una vez recolectados los enlaces, el script accede a cada URL de producto individual y extrae las imágenes de alta calidad disponibles mediante búsqueda de etiquetas ``. Para la descarga de imágenes, se utilizó la herramienta externa `aria2`, un gestor de descargas altamente eficiente que permite realizar múltiples conexiones simultáneas. Por último, se almacenaron las imágenes localmente. El dataset resultante consta de 1843 imágenes.

3.3. Preprocesamiento

Antes de proceder con el desarrollo de los modelos, se deben preparar las imágenes para garantizar su compatibilidad con *TensorFlow* y asegurar una estructura uniforme en cuanto a tamaño, formato y calidad.

Durante el proceso de descarga, algunas imágenes se almacenaron en formatos no estándar o con configuraciones de color no compatibles con *TensorFlow*. Para resolver este problema, se implementó una función que recorre todas las imágenes en la carpeta original, convierte cada una al espacio de color RGB y la guarda en formato .jpg. Esto permite estandarizar el conjunto de datos y evita errores durante la carga o procesamiento de los archivos. Además, se incluyó una verificación automática que detecta si el archivo es una imagen válida y si pertenece a uno de los formatos aceptados por *TensorFlow* (bmp, gif, jpeg, png). Las imágenes no compatibles fueron registradas y excluidas del conjunto final.

Una vez convertidas y validadas, las imágenes fueron redimensionadas a una resolución uniforme de 64×64 píxeles para reducir el coste computacional del entrenamiento. La resolución de 64×64 píxeles fue seleccionada por equilibrio entre calidad visual y viabilidad computacional. Resoluciones más altas podrían haber captado más detalles

en las prendas, pero requerían tiempos de entrenamiento y uso de memoria significativamente mayores, lo cual no era viable dentro de los recursos disponibles. Posteriormente, los valores de los píxeles fueron normalizados al rango $[-1, 1]$ dividiendo entre 127.5 y restando 1. Esta normalización es especialmente importante para los modelos que emplean activación \tanh , como las [GAN](#), para asegurar una correcta propagación del gradiente. A continuación, el dataset fue cargado en formato TensorFlow mediante la función `image_dataset_from_directory`, sin necesidad de etiquetas (`label_mode=None`), ya que el modelo generativo no requiere supervisión. Se utilizó un tamaño de batch de 128 y una interpolación del tipo `nearest`, adecuada para mantener los bordes y formas definidas en prendas de vestir.

Este proceso de preprocesamiento fue esencial para garantizar la calidad y coherencia del conjunto de datos antes del entrenamiento del modelo. Uno de los principales retos durante el preprocesamiento fue la homogeneización de un conjunto de datos visualmente heterogéneo. Las imágenes de pasarela presentan fondos complejos y variaciones de iluminación, mientras que las imágenes de catálogo son mucho más uniformes. Esta diferencia afectó posteriormente a la calidad del aprendizaje en ciertos modelos.

Las imágenes tenían el formato mostrado en la figura 3.2.



Figura 3.2. Imágenes del dataset después del preprocesamiento

3.4. DCGAN con 4 capas convolucionales

El generador está diseñado para transformar un vector aleatorio de dimensión 100 en una imagen RGB de tamaño 64×64 . La red comienza con una capa densa que proyecta el vector latente a un volumen tridimensional inicial $4 \times 4 \times 1024$. A continuación, se aplican cuatro capas Conv2DTranspose, que duplican progresivamente la altura y anchura de la imagen mientras disminuyen la profundidad. La secuencia de canales es la siguiente: $1024 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 3$. En cada capa se utiliza un tamaño de *kernel* de 5×5 , *strides*=(2, 2) para duplicar las dimensiones espaciales, y *padding*=*'same'* para mantener la alineación de los bordes. Además, se establece el parámetro *use_bias=False*, lo que indica que no se añade un término de sesgo explícito en las capas convolucionales. Esto se hace porque el uso combinado de Batch-Normalization y ReLU en cada bloque ya permite centrar y escalar adecuadamente las activaciones, haciendo innecesario el sesgo adicional.

La última capa emplea una activación tanh para que los valores de los píxeles de salida estén en el rango $[-1, 1]$, que coincide con la normalización aplicada a las imágenes reales.

Si generamos una muestra de ruido aleatorio (Figura 3.3) y se la pasamos al generador antes del entrenamiento obtenemos la siguiente imagen sintética.

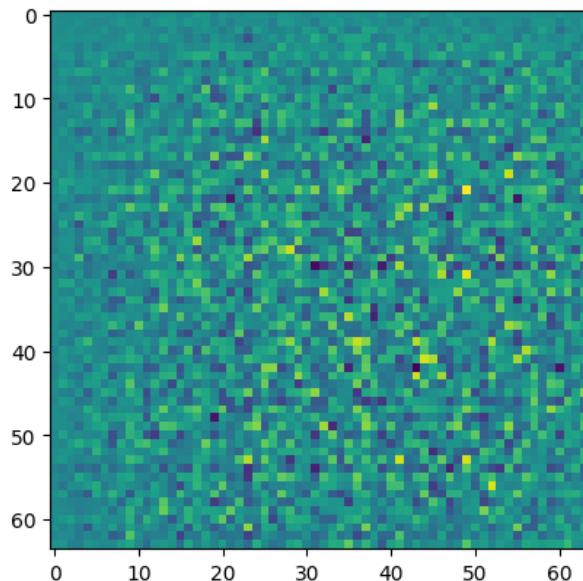


Figura 3.3. Ejemplo de imagen sintética generada por el generador

El discriminador toma como entrada una imagen de tamaño $64 \times 64 \times 3$ (RGB) y devuelve una puntuación escalar que indica si la imagen es real o generada. La estructura del discriminador está compuesta por tres capas convolucionales Conv2D con `strides=(2, 2)` y `padding='same'`, que reducen progresivamente la resolución espacial de la imagen (de 64×64 a 8×8) mientras aumentan la profundidad de los filtros: $[128 \rightarrow 256 \rightarrow 512]$.

Se utiliza la activación LeakyReLU en lugar de ReLU para evitar el problema de *dead neurons*, es decir, unidades que dejan de activarse durante el entrenamiento. Además, se emplea *Dropout* con una tasa de 0.3 en cada capa convolucional para reducir el sobreajuste mediante la introducción de ruido durante el proceso de entrenamiento.

Después de las capas convolucionales, se aplica una operación de aplanamiento (`Flatten`) para convertir el volumen tridimensional en un vector unidimensional, seguido de una capa densa con una única neurona, que devuelve la puntuación final del discriminador.

Si pasamos la imagen generada anteriormente por el generador antes del entrenador, el discriminador produce una salida de un valor cercano a cero ya que consigue distinguir que es una imagen falsa.

Para entrenar la red adversaria, se utiliza la función de pérdida `BinaryCrossentropy` incluida en TensorFlow. Esta función mide la diferencia entre las predicciones del discriminador y las etiquetas reales (1 para imágenes reales, 0 para generadas). Se han definido dos funciones separadas de pérdida para el generador y el discriminador:

- La pérdida del discriminador se compone de dos términos: uno para las imágenes reales (debe predecir 1) y otro para las imágenes falsas (debe predecir 0). La suma de ambos constituye la pérdida total del discriminador.
- La pérdida del generador se define con respecto a las predicciones del discriminador sobre las imágenes generadas. Como el objetivo del generador es “engañar” al discriminador, se desea que este prediga 1 para las imágenes falsas. Por tanto, se minimiza la diferencia entre las predicciones del discriminador y la etiqueta 1.

Durante el entrenamiento se genera un lote de ruido aleatorio que sirve como entrada al generador. Este produce un conjunto de imágenes sintéticas que se pasan, junto con imágenes reales, al discriminador. Ambas redes se entrenan simultáneamente mediante el uso de dos objetos `GradientTape` de TensorFlow, que permiten calcular y aplicar los gradientes correspondientes a sus respectivas funciones de pérdida.

Cada iteración de entrenamiento procesa un *batch* de imágenes, que corresponde a un subconjunto del conjunto de datos. El uso de batches permite actualizar los pesos de la red de forma más eficiente que si se entrenara con todo el conjunto de datos de una sola vez, además de ayudar a estabilizar el gradiente.

El bucle de entrenamiento completo recorre el conjunto de datos durante un número determinado de (*epochs*). Tras cada época, se genera un conjunto de imágenes a partir de una semilla fija, lo que permite observar de forma consistente la evolución del generador a lo largo del tiempo. Esta semilla (*seed*) se define al comienzo del entrenamiento como una matriz de ruido aleatorio de dimensión `[num_examples_to_generate, noise_dim]`. Utilizar una semilla constante permite comparar las imágenes generadas en distintas épocas con el mismo punto de partida, facilitando la evaluación visual del progreso del modelo.

Para visualizar los resultados del generador se ha desarrollado una función que toma como entrada el modelo, la época actual y la semilla, y genera una cuadrícula de imágenes que se guardan como archivos PNG. Para representar correctamente las imágenes, se reescalan desde el rango $[-1, 1]$ (salida del generador con activación tanh) al rango $[0, 1]$, compatible con funciones de visualización como `imshow` de `matplotlib`.

El modelo fue entrenado durante 100 épocas con un tamaño de batch de 128, utilizando el optimizador Adam con una tasa de aprendizaje $\alpha = 0,0002$, $\beta_1 = 0,5$ y $\beta_2 = 0,999$. Estos valores son estándar en la literatura para estabilizar el entrenamiento de GANs y se mantuvieron constantes a lo largo de los experimentos.

3.5. DCGAN con 4 capas convolucionales y data augmentation

Hasta ahora se ha utilizado el dataset original. Sin embargo, el reducido tamaño de este nos lleva a plantearnos el uso de técnicas de *data augmentation* para generar un conjunto de datos más variado. Esta técnica consiste en aplicar transformaciones como rotaciones, giros, zooms, ajuste de luces y colores, etc. con el fin de obtener un dataset con más heterogéneo y con una mayor variedad de enfoques, de modo que el modelo sea capaz de aprender mejor las características de las imágenes y desarrollar una representación más robusta frente a variaciones en los datos de entrada.

Para su implementación, se utilizó el módulo `tf.keras.layers` de TensorFlow, construyendo un pipeline de transformaciones compuesto por las siguientes operaciones:

- **Giros horizontales aleatorios:** con probabilidad 0.5, para simular simetrías presentes en prendas como chaquetas, vestidos o camisetas.
- **Rotaciones discretas:** de hasta 15 grados, aplicadas mediante combinaciones de rotaciones en múltiplos de 90 grados para mantener alineación estructural.
- **Ajustes de brillo y contraste:** pequeñas variaciones controladas para simular condiciones de iluminación no homogéneas.
- **Recorte aleatorio seguido de redimensionamiento:** recortes parciales en diferentes zonas de la imagen para exponer al modelo a casos con prendas parcialmente visibles. Posteriormente, las imágenes se reescalaron a 64×64 píxeles para mantener compatibilidad con la arquitectura de la DCGAN.

Estas operaciones se aplicaron en tiempo real durante la carga de los datos mediante la API de TensorFlow, lo cual permitió ahorrar espacio de almacenamiento y garantizar que cada epoch recibiera un conjunto distinto de ejemplos. El dataset final después de aplicar *data augmentation* queda como en la figura 3.4.

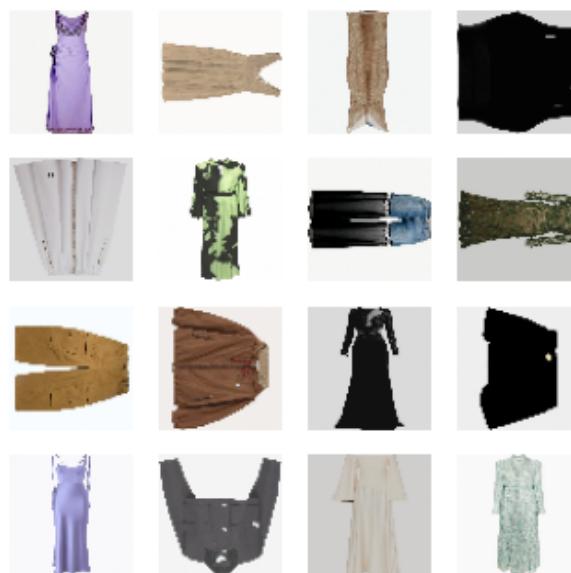


Figura 3.4. Muestras del nuevo dataset después de aplicar técnicas de data augmentation

El modelo [DCGAN](#) utilizado en esta variante mantiene la misma arquitectura descrita previamente, pero se entrena sobre el nuevo conjunto aumentado. Este enfoque permite evaluar el impacto directo del *data augmentation* sobre la calidad visual y diversidad de las imágenes generadas. Tal como se discute en la sección de resultados, esta técnica mejora considerablemente la estabilidad del entrenamiento y favorece la aparición de patrones estructurados en las muestras sintetizadas.

3.6. DCGAN con 6 capas convolucionales y data augmentation

En esta segunda implementación, se ha mantenido la estructura básica de la [DCGAN](#), pero se han introducido ciertas modificaciones tanto en la arquitectura de los modelos como en la función de pérdida, con el objetivo de mejorar la estabilidad del entrenamiento y la calidad de las imágenes generadas. Además, se ha utilizado un conjunto de datos ampliado mediante técnicas de *data augmentation*.

La primera capa densa del generador ha sido modificada para proyectar el vector latente en un volumen tridimensional inicial de tamaño $8 \times 8 \times 1024$, en lugar de $4 \times 4 \times 1024$. Esto permite una mayor resolución inicial, facilitando la generación de detalles espaciales más coherentes en las capas posteriores. Además, se han añadido dos nuevas capa convolucionales de 1024 y 512 filtros y sus respectivas capas de BatchNormalization y funciones de activación LeakyReLu. Por tanto, la nueva secuencia de los canales es $[1024 \rightarrow 512 \rightarrow 512 \rightarrow 256 \rightarrow 128 \rightarrow 64]$.

El discriminador se ha simplificado al reducir el número de filtros en cada capa convolucional para evitar un desequilibrio en la potencia del modelo con respecto al generador. En esta configuración, los canales siguen la secuencia: $[64 \rightarrow 128 \rightarrow 256]$, lo cual reduce la complejidad del modelo y, en consecuencia, puede favorecer una convergencia más estable durante el entrenamiento.

Otra diferencia importante se encuentra en la función de pérdida del discriminador. Se ha introducido un coeficiente multiplicativo de 0.9 sobre la pérdida correspondiente a las imágenes reales:

```
real_loss = cross_entropy(tf.ones_like(real_output), real_output)
```

* 0.9

Este ajuste actúa como una forma de *label smoothing*, técnica que consiste en suavizar las etiquetas reales para evitar que el discriminador se vuelva demasiado confiado y dominante respecto al generador. Esta estrategia favorece un equilibrio más estable en la dinámica adversaria, lo que puede resultar en una mejor calidad de las imágenes generadas.

3.7. VAE con 3 capas convolucionales con data augmentation

En base a las observaciones realizadas durante los primeros experimentos, se comprobó que los modelos generativos obtenían resultados notablemente mejores cuando se entrenaban sobre datos aumentados artificialmente. Por este motivo, se incorporó un pipeline de *data augmentation* aplicado dinámicamente durante el entrenamiento. Las transformaciones incluyeron giros horizontales, rotaciones aleatorias de hasta 15 grados, pequeñas traslaciones, ajustes de brillo, contraste y saturación, así como una normalización final al rango $[-1, 1]$ mediante *Rescaling*. Estas operaciones tienen como objetivo aumentar la variabilidad visual del conjunto de entrenamiento sin alterar su semántica, lo que fuerza al modelo a aprender representaciones más robustas y generalizables. Para el conjunto de validación se aplicó únicamente un redimensionamiento y la misma normalización, asegurando una evaluación consistente del desempeño sin introducir ruido artificial.

Para esta aproximación se ha implementado un Autoencoder Variacional Convolucional, adaptado al tamaño de imagen 64×64 y diseñado con una arquitectura compacta que permite explorar el comportamiento de los modelos generativos probabilísticos en contextos de baja dimensión latente.

El modelo se compone de dos bloques principales: un codificador (encoder) y un decodificador (decoder), ambos construidos con capas convolucionales. El codificador toma como entrada un tensor de tamaño $64 \times 64 \times 3$, al que se le aplican dos capas convolucionales con 32 y 64 filtros, respectivamente, con kernel de 3×3 , *strides* de $(2, 2)$, y activación ReLU. Estas capas reducen progresivamente la resolución espacial al tiempo que incrementan la profundidad del tensor, extrayendo representaciones relevantes de

las imágenes de entrada.

A la salida del codificador, las activaciones se aplana y se proyectan a un vector de dimensión $2 \cdot z$, donde z es la dimensión del espacio latente. En esta implementación se ha fijado $z = 2$ con el fin de facilitar la visualización bidimensional de las distribuciones latentes. El vector proyectado se interpreta como la concatenación de la media μ y la desviación estándar logarítmica $\log \sigma^2$ de una distribución gaussiana multivariante. Esta parametrización permite aplicar el método de reparametrización, que genera muestras latentes z mediante la fórmula:

$$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

Este mecanismo hace posible que el modelo sea completamente diferenciable, permitiendo su entrenamiento mediante retropropagación.

El decodificador toma una muestra del espacio latente y genera una imagen RGB reconstruida. Para ello, primero transforma el vector latente en un volumen tridimensional de tamaño $8 \times 8 \times 32$ mediante una capa densa, seguido de una operación de `reshape`. A continuación, se aplican tres capas convolucionales transpuestas con 64, 32 y 3 filtros respectivamente. Las dos primeras utilizan activación ReLU, mientras que la última emplea una función `sigmoid` para restringir los valores de salida al rango $[0, 1]$, adecuado para representar intensidades de píxel.

El modelo se entrena maximizando la cota inferior de la evidencia ([ELBO](#)), que combina dos términos: un error de reconstrucción y un término de regularización. El primero mide la discrepancia entre la imagen de entrada y la imagen reconstruida; el segundo corresponde a la divergencia de Kullback-Leibler entre la distribución latente aproximada $q_\phi(z|x)$ y una distribución gaussiana estándar $\mathcal{N}(0, I)$.

El modelo fue entrenado utilizando el optimizador Adam con una tasa de aprendizaje de 1×10^{-4} . Gracias al uso del método de reparametrización, tanto el codificador como el decodificador se optimizan de forma conjunta a través del cálculo de gradientes. Esta arquitectura sencilla pero funcional permite evaluar el potencial del VAE para generar imágenes coherentes de prendas de ropa, así como explorar las propiedades geométricas del espacio latente.

3.8. VAE con arquitectura convolucional profunda y data augmentation

En esta segunda implementación, se ha optado por una arquitectura más profunda y regularizada, con el objetivo de mejorar la capacidad de representación del modelo y generar imágenes más detalladas y coherentes.

Al igual que en los modelos anteriores se ha llevado a cabo un proceso de preprocesamiento para aplicar *data augmentation*. El pipeline de transformaciones es igual al mencionado en los otros apartados.

La principal diferencia respecto a la versión anterior es la incorporación de más capas convolucionales tanto en el codificador como en el decodificador, así como el uso sistemático de técnicas de normalización y regularización. Concretamente, cada bloque convolucional está compuesto por una secuencia de capas que incluye: normalización por lotes (BatchNormalization) para estabilizar las activaciones, la función de activación ReLU para introducir no linealidad, y una capa de Dropout con tasa 0.3 para prevenir el sobreajuste.

El codificador recibe como entrada un tensor de tamaño $64 \times 64 \times 3$ (imagen RGB) y aplica tres capas convolucionales 2D con 32, 64 y 128 filtros, respectivamente. Cada capa tiene un kernel de 3×3 , strides de $(2, 2)$ y padding='same', lo que permite reducir progresivamente la resolución espacial a la vez que se aumenta la profundidad de las representaciones. A continuación, se realiza una operación de aplanado y se proyecta la salida en un vector de dimensión $2 \cdot z$, donde z es la dimensión del espacio latente. Al igual que en el primer modelo, esta salida representa la concatenación de la media y la desviación estándar logarítmica de la distribución latente.

El decodificador toma como entrada una muestra del espacio latente y la transforma en una imagen sintética de salida. Primero se expande el vector latente con una capa densa a un volumen tridimensional de tamaño $8 \times 8 \times 128$. Este volumen inicial se reconstruye a través de cuatro capas convolucionales transpuestas con 128, 64, 32 y 3 filtros respectivamente. Las tres primeras capas utilizan activación ReLU, y la última capa aplica una función de activación sigmoid para garantizar que la imagen generada tenga valores de píxel en el rango $[0, 1]$. Esta configuración permite generar imágenes RGB con formato probabilístico, adecuado para la función de pérdida basada en entropía cruzada.

Gracias a esta arquitectura más profunda y regularizada, el modelo puede aprender representaciones latentes más estructuradas y generar imágenes con mayor calidad visual. En la sección de resultados se analizan los beneficios concretos de esta arquitectura frente a versiones más simples, evaluando tanto la coherencia estructural como la diversidad de las muestras generadas.

3.9. Modelo de Difusión

El desarrollo de este modelo se ha llevado a cabo mediante el uso de Pytorch. Se han implementado en diversas funciones las distintas etapas de un modelo de difusión: proceso de difusión y la posterior eliminación del ruido para la generación de nuevas muestras.

Para el modelo de difusión, el preprocesamiento de las imágenes se ha realizado utilizando la librería `torchvision` del ecosistema PyTorch, en lugar del flujo de datos de `TensorFlow` utilizado en implementaciones anteriores. En este caso, se define una transformación compuesta (`transforms.Compose`) que incluye una serie de operaciones aplicadas secuencialmente a cada imagen durante su carga. Primero, todas las imágenes se redimensionan a 64×64 píxeles para garantizar una entrada consistente en el modelo. A continuación, se aplican transformaciones de aumento de datos similares a las utilizadas en modelos anteriores: giros horizontales aleatorios, rotaciones de hasta 15 grados, desplazamientos (`affine translate`) de hasta un 5 % y variaciones en el color mediante la clase `ColorJitter`, que modifica aleatoriamente el brillo, contraste, saturación y tono de la imagen. Finalmente, las imágenes se convierten a tensores con `ToTensor()` y se normalizan al rango $[-1, 1]$ mediante `Normalize`. Esta normalización es necesaria para garantizar una propagación estable del ruido durante las etapas de degradación y reconstrucción del modelo de difusión. La combinación de estas operaciones permite alimentar el modelo con ejemplos variados y bien escalados, lo que mejora la robustez del entrenamiento y la capacidad de generalización.

El proceso de difusión difumina las imágenes añadiendo ruido gaussiano en T intervalos. Se comienza con un vector de ruido β_t linealmente espaciado entre un valor inicial y uno final. A partir de estos valores se obtienen los coeficientes $\alpha_t = 1 - \beta_t$, y su producto acumulado $\bar{\alpha}_t = \prod s = 1^t \alpha_s$. En cada paso t , se genera una imagen ruidosa x_t a partir de la imagen original x_0 mediante la ecuación 2.6.

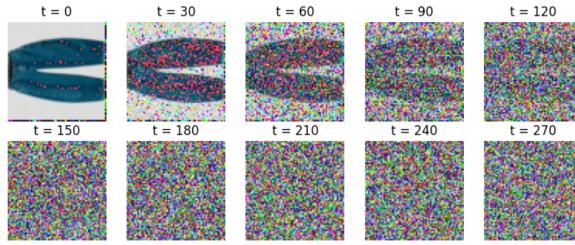


Figura 3.5. Evolución de una imagen durante el proceso de difusión

La imagen 3.5 muestra cómo la imagen original se desenfoca progresivamente mediante la adición de ruido gausiano a lo largo de 300 pasos.

Ahora, hay que implementar el modelo inverso, reconstruir la imagen del paso anterior dado el estado actual. El proceso inverso se inicia con una imagen de ruido puro $x_T \sim \mathcal{N}(0, I)$, y en cada paso t se calcula el valor de x_{t-1} resolviendo la ecuación 2.8. Este proceso se repite desde $t = T$ hasta $t = 0$ para generar una nueva muestra sintética.

El modelo elegido encargado de aprender el proceso inverso es una arquitectura tipo **U-Net** [30], modificada para incluir información del paso temporal t mediante un embedding posicional sinusoidal. Este vector temporal se incorpora a cada bloque del modelo mediante una transformación lineal y una suma a los mapas de activación.

La arquitectura **U-Net** está conformada por bloques down para la compresión y bloques up para la reconstrucción. En nuestro caso, hemos usado la siguiente secuencia de filtros $[64 \rightarrow 128 \rightarrow 256 \rightarrow 512 \rightarrow 1024]$ para los bloques down y la secuencia inversa para los bloques up. Además, después de cada bloque se aplica convolución y los vectores temporales.

A partir de esa media, se genera la imagen del paso anterior añadiendo ruido con la varianza posterior. Este proceso se repite desde $t = T$ hasta $t = 0$ para generar una nueva muestra sintética.

El modelo se entrena para predecir el ruido ε añadido en el paso de difusión. Para ello, se genera una imagen ruidosa x_t y se entrena la red para aproximar el ruido original a partir de x_t y del paso t . La función de pérdida empleada es la pérdida L1 entre el ruido real y el ruido predicho. Se utiliza el optimizador Adam con una tasa de aprendizaje de 0.0001, y se entrena durante 100 épocas con un tamaño de lote de 32 imágenes.

La implementación del modelo se ha desarrollado íntegramente en PyTorch. A continuación se detallan los componentes principales del pipeline y su funcionamiento.

Para simular el proceso de difusión, se definen las siguientes variables clave:

- β_t : secuencia lineal de varianzas que controla la cantidad de ruido añadida en cada paso.
- $\alpha_t = 1 - \beta_t$: coeficiente que conserva la señal original.
- $\bar{\alpha}_t = \prod_{s=1}^t \alpha_s$: producto acumulado que representa la proporción de la imagen original preservada en el paso t .
- t : paso de tiempo actual del proceso de difusión, muestreado aleatoriamente durante el entrenamiento.

Estas variables permiten construir una planificación del ruido que degrada progresivamente una imagen real hacia ruido gaussiano.

Mediante una función llamada `forward_noise` implementa el proceso directo de difusión. A partir de una imagen \mathbf{x}_0 , genera dos versiones ruidosas: \mathbf{x}_t y \mathbf{x}_{t+1} , utilizando la misma realización de ruido ϵ . Esto se expresa matemáticamente como:

$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \cdot \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \epsilon$$

Esta formulación permite entrenar el modelo para aprender cómo evoluciona la imagen a medida que se le añade ruido.

El modelo adoptado sigue una arquitectura inspirada en **U-Net**, adaptada para tareas de predicción de ruido en imágenes. Esta red está compuesta por bloques convolucionales residuales que incluyen la codificación explícita del paso de tiempo. Cada bloque recibe dos entradas:

- Una imagen ruidosa \mathbf{x}_t
- Un vector embebido del tiempo t , que modula los canales activando partes distintas de la red dependiendo del nivel de ruido

Cada bloque aplica una combinación de convoluciones, normalización y multiplicación por el embedding temporal para condicionar la salida en función del ruido presente.

Durante la inferencia, las predicciones del modelo se realizan paso a paso, comenzando desde una muestra de ruido gaussiano $\mathbf{x}_T \sim \mathcal{N}(0, I)$. En cada paso t , se utiliza la predicción del ruido $\varepsilon_\theta(\mathbf{x}_t, t)$ para calcular la imagen menos ruidosa \mathbf{x}_{t-1} mediante:

$$\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \cdot \varepsilon_\theta(\mathbf{x}_t, t) \right) + \sigma_t \cdot \mathbf{z}$$

Se ha implementado una función adicional que permite visualizar el proceso de denoising paso a paso, mostrando cómo el modelo transforma progresivamente una muestra de ruido en una imagen coherente.

Para entrenar el modelo, se generan pares de imágenes $(\mathbf{x}_t, \mathbf{x}_{t+1})$ utilizando la función de difusión. El modelo recibe como entrada \mathbf{x}_t junto con el tiempo t , y se entrena para predecir \mathbf{x}_{t+1} , es decir, la imagen con un nivel de ruido ligeramente superior. La función de pérdida utilizada combina el error L1 y el **MSE**:

$$\mathcal{L} = 0,9 \cdot \text{L1}(y_{\text{pred}}, \mathbf{x}_{t+1}) + 0,1 \cdot \text{MSE}(y_{\text{pred}}, \mathbf{x}_{t+1})$$

Este enfoque busca estabilizar el entrenamiento utilizando la robustez del error absoluto y la precisión del error cuadrático.

4.

Resultados

4.1. Presentación de los resultados

La [DCGAN](#) con 4 capas convolucionales se entrenó durante 20 epochs. Como vemos en la imagen [4.1](#), los resultados no fueron muy prometedores. Las nuevas imágenes generadas apenas consiguen detectar o identificar ningún patrón, silueta o característica visual.

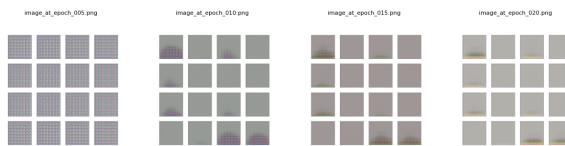


Figura 4.1. Imágenes generadas a lo largo de 20 epochs

Esta falta de capacidad para capturar adecuadamente las relaciones y características intrínsecas de los datos, lo que impide una generación eficiente de nuevas muestras, puede deberse al escaso número de *epochs*. Sin embargo, el comportamiento de las pérdidas tanto del generador como del discriminador sugiere que la **arquitectura** de ambos modelos podría **no ser la más adecuada**, lo que limitaría su capacidad para aprender representaciones más complejas y detalladas.

Tabla 4.1. Pérdidas de las primeras y últimas 3 épocas

Epoch	Gen Loss	Disc Loss
1	1.7522	0.3246
2	2.6605	0.1098
3	3.3096	0.0726
18	8.9887	0.0637
19	7.3223	0.5032
20	10.2477	0.0266

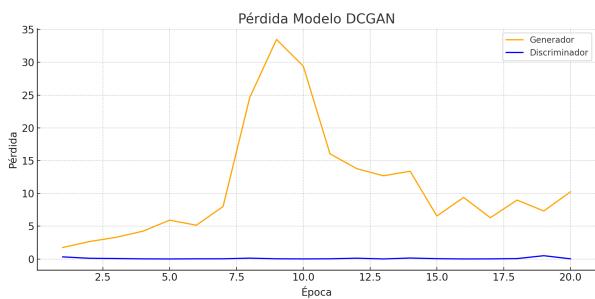


Figura 4.2. Gráfica de pérdidas del entrenamiento

La pérdida del generador que vemos en las figuras 4.1 y 4.2 aumenta muy rápido y la del discriminador disminuye demasiado pronto. Esto sugiere varios aspectos:

- El discriminador aprende **demasiado rápido** a distinguir las imágenes reales de las sintéticas.
- El discriminador se vuelve **muy fuerte** y mucho más potente que el generador, lo que indica un **fuerte desequilibrio** en ambas arquitecturas.
- El generador no es capaz de generar **imágenes convincentes** y no consigue engañar al discriminador, con lo que su pérdida sigue aumentando.
- Si la pérdida del generador sigue aumentando o no consigue alcanzar un **equilibrio** con el discriminador, entonces **no se está produciendo un buen aprendizaje** y está atascado.

4.1.1. DCGAN con 4 capas convolucionales y data augmentation

El modelo DCGAN con 4 capas y *data augmentation* presenta la **misma arquitectura** que en el modelo anterior, la única diferencia es que este modelo se ha entrenado con un dataset sobre el cual se han aplicado técnicas de *data augmentation* y el entrenamiento ha sido durante 100 epochs. En esta ocasión se pueden observar mejoras considerables en las imágenes generadas con respecto al modelo anterior (Figura 4.3).

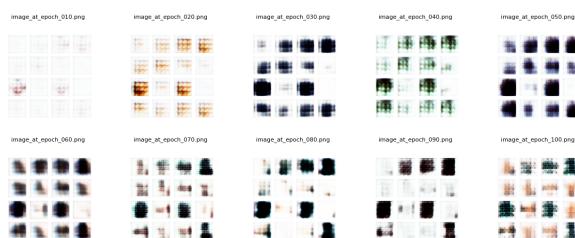


Figura 4.3. Imágenes generadas cada 10 epochs por el modelo DCGAN básico

Durante las 30 primeras *epochs* solo se genera ruido o imágenes borrosas que no representan nada. No es hasta la *epoch* 30 que se pueden identificar algunas formas básicas que, lejos de representar una prenda, suponen un avance con respecto a los resultados anteriores. Asimismo, los colores en las muestras consiguen integrarse con otros de forma que en los resultados de la época 100 presentan **mayor variedad de colores y siluetas**. A pesar de que los resultados no sean satisfactorios, se pueden apreciar las

mejoras simplemente con la integración de esta técnica al conjunto de datos y sin haber modificado la arquitectura.

Tabla 4.2. Pérdidas: primeras y últimas 3 épocas

Epoch	Gen Loss	Disc Loss
1	0.6644	0.6644
2	0.6664	1.4188
3	0.7092	1.3651
98	0.7337	1.4350
99	0.6877	1.2147
100	0.7321	1.1441

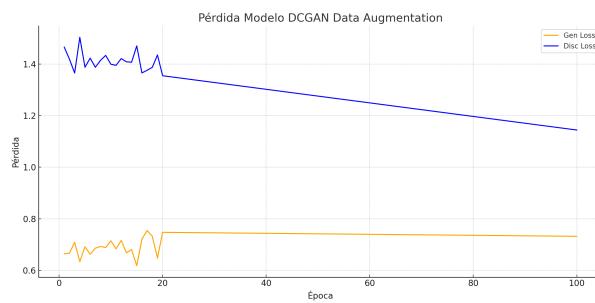


Figura 4.4. Gráfica de pérdidas del entrenamiento de la DCGAN con data augmentation

Las gráficas 4.2 y 4.4 presentan unos **valores más equilibrados** entre ambas pérdidas. A diferencia de la anterior, en la que la diferencia entre el generador y el discriminador era muy marcada, aquí se observa que con el transcurso de las *epochs* la pérdida del generador se mantiene prácticamente constante a partir de la *epoch* número 20, pero logrando cierta estabilidad. Por su parte, la pérdida del discriminador presenta una disminución progresiva lo que se traduce en el aumento de la dificultad para discernir entre las imágenes sintéticas y las reales. En conclusión, se aprecia que la gráfica del generador tiende a mantenerse constante mientras que la del discriminador aumenta poco a poco. Esto indica que el proceso de entrenamiento es el adecuado y que, de continuar con el entrenamiento, se podrían conseguir unos resultados más realistas y coherentes.

Esta mejora se debe a la introducción de técnicas de *data augmentation* en el conjunto de datos. Al generar ejemplos con mayor variedad en cuanto a orientación, iluminación, color y composición, se enriquece la diversidad del dataset, lo que obliga a ambos modelos —especialmente al discriminador— a **generalizar mejor** y no memorizar patrones específicos. Como consecuencia, el generador es capaz de aprender **representaciones más robustas y consistentes**, lo que se traduce en resultados visuales más realistas y en un entrenamiento más equilibrado y efectivo.

4.1.2. DCGAN con 6 capas convolucionales

Esta última implementación incluye más capas convolucionales para capturar de forma más eficaz las características, pero los resultados no experimentan mejoras sustanciales.

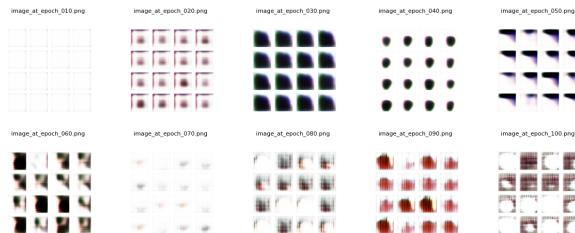


Figura 4.5. Evolución de las imágenes generadas por este modelo a lo largo de las 100 epochs

Tal y como se muestra en la figura 4.5, las primeras 10 *epochs* obtenemos resultados vacíos, esto es normal ya que en las primeras épocas la red todavía no ha conseguido aprender ningún patrón significativo. A partir de la 20, aparecen por primera vez ciertos patrones que, aunque repetitivos, poseen cierta estructura simétrica. Estos patrones se mantienen **repetitivos e incompletos**, pero experimentan ganancias en la calidad del color, definición y textura.

Tabla 4.3. Pérdidas: primeras y últimas 3 epochs

Epoch	Gen Loss	Disc Loss
1	0.6170	1.3899
2	0.9149	1.2089
3	0.7593	1.4229
98	0.7181	1.2970
99	0.6617	1.4092
100	0.7681	1.2975

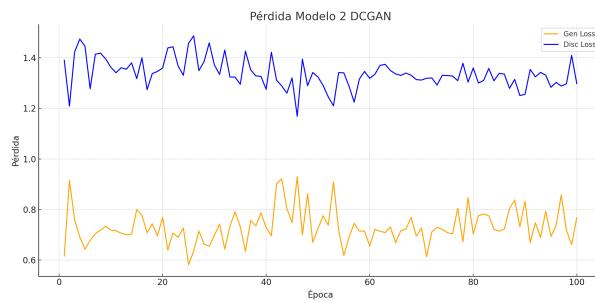


Figura 4.6. Gráfica de pérdidas del entrenamiento del modelo 2 DCGAN

Aunque los resultados no sean demasiado buenos, las gráficas de las medidas de pérdida 4.3 y 4.6 presenta una mejor estructura. El valor de pérdida del generador varía mucho más pero siempre oscila en valores entre 0,6 y 0,8. El valor de pérdida del discriminador no disminuye progresivamente sino que se mantiene sobre un valor cercano a 0,4. Estos resultados son esperanzadores debido a que es una señal de que si entrenáramos durante más *epochs* conseguiríamos mejores resultados.

4.1.3. VAE con 3 capas convolucionales con data augmentation

En su gran mayoría, los modelos [Variational Autoencoders](#) no son particularmente eficaces en la generación de imágenes muy reales, precisas y cargadas de detalles visuales. A pesar de ello, son muy útiles porque consiguen generar imágenes nuevas y más comprimidas a partir de las características intrínsecas de las imágenes originales.

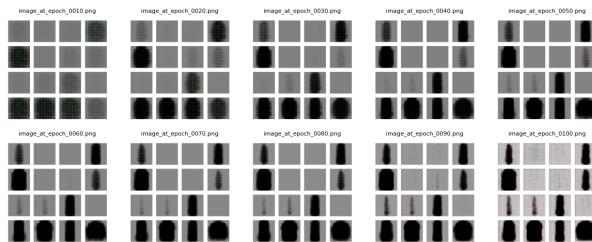


Figura 4.7. Imágenes generadas por el modelo Variational Autoencoder con una arquitectura básica

La figura 4.7 muestra los resultados de las nuevas muestras generadas. Un detalle remarcable es que aprende relativamente pronto puesto que ya en la epoch 10 ha retenido cierta información sobre las siluetas. A lo largo del proceso, las "manchas iniciales adoptan definición y concreción. Así, en la última iteración se ven claramente **figuras y siluetas** correspondientes a prendas de ropa. El punto negativo es que este modelo de aprendizaje no es especialmente bueno reteniendo información sobre los colores. Al principio se ve que genera el fondo gris y manchas de colores, pero conforme aprendía a generar el blanco del fondo, el resto de colores también perdían importancia generando diseños en negro.

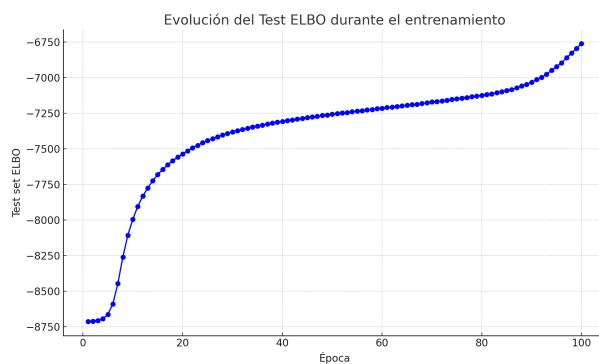


Figura 4.8. Evolución de la métrica ELBO en el primer modelo Variational Autoencoder

La gráfica 4.8 de la evolución de la métrica **ELBO** presenta una forma ascendente, es decir, cada vez se va haciendo menos negativa. Esto es bueno porque indica que el

modelo está optimizando la evidencia inferior, logrando una mejor aproximación a la distribución verdadera de los datos. En otras palabras, el modelo está aprendiendo a reconstruir las imágenes con mayor precisión.

4.1.4. VAE con arquitectura convolucional profunda y data augmentation

En este modelo hemos introducido una arquitectura con más capas convolucionales, regularización y funciones de activación.

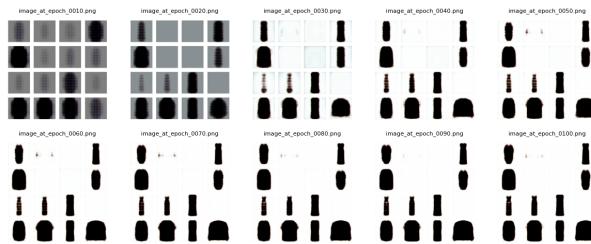


Figura 4.9. Imágenes generadas por el modelo Variational Autoencoder con una arquitectura más compleja

Los resultados generados por este nuevo modelo que se muestran en 4.9 son muy similares visualmente a los generados con el anterior. Aunque se puede observar que las siluetas están mínimamente más definidas y son más precisas, el problema del color sigue presente.

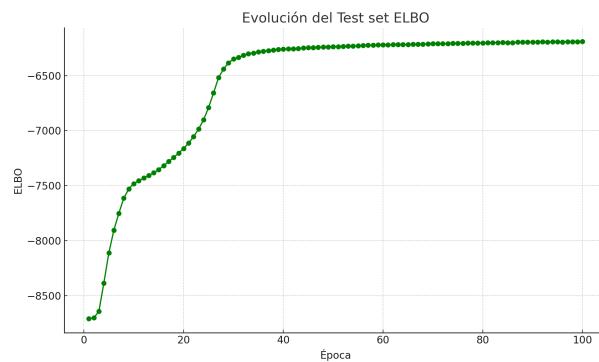


Figura 4.10. Evolución de la métrica ELBO en un Variational Autoencoder con una arquitectura más compleja

En este caso, la gráfica 4.10 nos da información positiva. Los valores tienden a hacerse más positivos completando un buen entrenamiento. En este caso, los valores tienden a

hacerse más positivos con más rapidez obteniendo en el mismo número de *epochs* un mejor valor en el test de [ELBO](#) (-6759.73 vs -6189.18).

4.1.5. Modelo de Difusión

El caso de los modelos de difusión es similar al de los [Variational Autoencoders](#). En nuestra implementación mostramos el proceso de *denoising* de la imagen original y cómo se llega desde la imagen ruidosa a la nueva muestra generada.

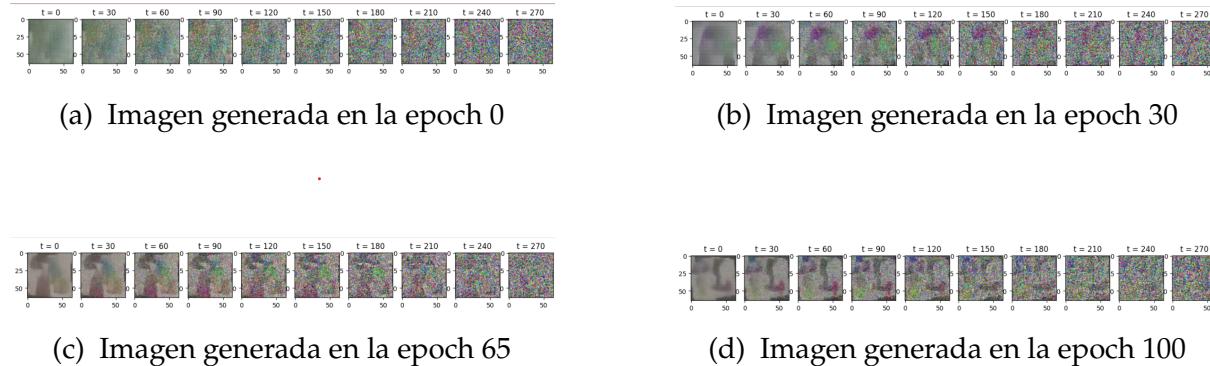


Figura 4.11. Proceso de generación de imágenes durante T en las epochs 0, 30, 65 y 100

Los resultados de la imagen [4.11](#) todavía no muestran figuras ni formas reconocibles. No obstante, se puede apreciar en cada época representada la mejora en la capacidad de generación de nuevas imágenes a partir de ruido. Conforme se completan las *epochs* los resultados producidos son más concretos, captan más características visuales como el color y generando formas, aunque todavía no se haya alcanzado ninguna que sea reconocible.

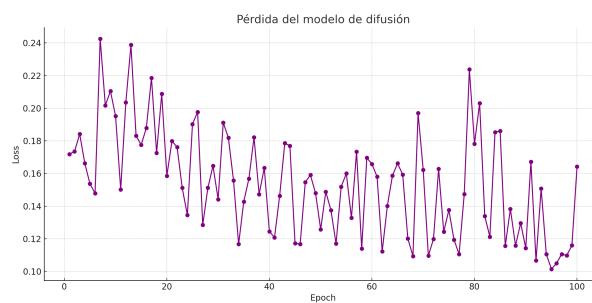


Figura 4.12. Gráfica de la pérdida del modelo de difusión

Esta gráfica 4.12 presenta cómo evoluciona la función de pérdida, en nuestro caso L1, que mide la diferencia entre el ruido generado y la distribución del ruido objetivo. Se observa una **tendencia descendente** a lo largo de las épocas de entrenamiento, lo cual indica que el modelo está aprendiendo adecuadamente a aproximar la distribución de ruido esperada. Este comportamiento sugiere que el entrenamiento ha sido exitoso y que el modelo está aprendiendo correctamente a generar nuevas muestras visualmente coherentes. Seguramente, si entrenásemos el modelo durante más *epochs* obtendríamos resultados con mayor calidad visual.

4.2. Comparación global de los modelos

Para facilitar la comparación entre los diferentes enfoques propuestos, en esta sección se presentan los resultados generados por cada modelo en condiciones equivalentes. Debido a la variedad de técnicas y parámetros utilizados, vamos a incluir gráficas que comparan los distintos modelos en función de dos características: el uso de data augmentation y el tipo de arquitectura (básica/profunda). El primer modelo de DCGAN sin data augmentation no se mostrará ya que los parámetros y condiciones de entrenamiento son muy diferentes al resto. Las figuras muestran una selección de imágenes generadas en la última época de entrenamiento para cada uno de los modelos implementados.

La figura 4.13 muestra los resultados en la época 100 por los tres modelos entrenados con una arquitectura más simple, es decir, usando menos capas convolucionales y aplicando *data augmentation*.

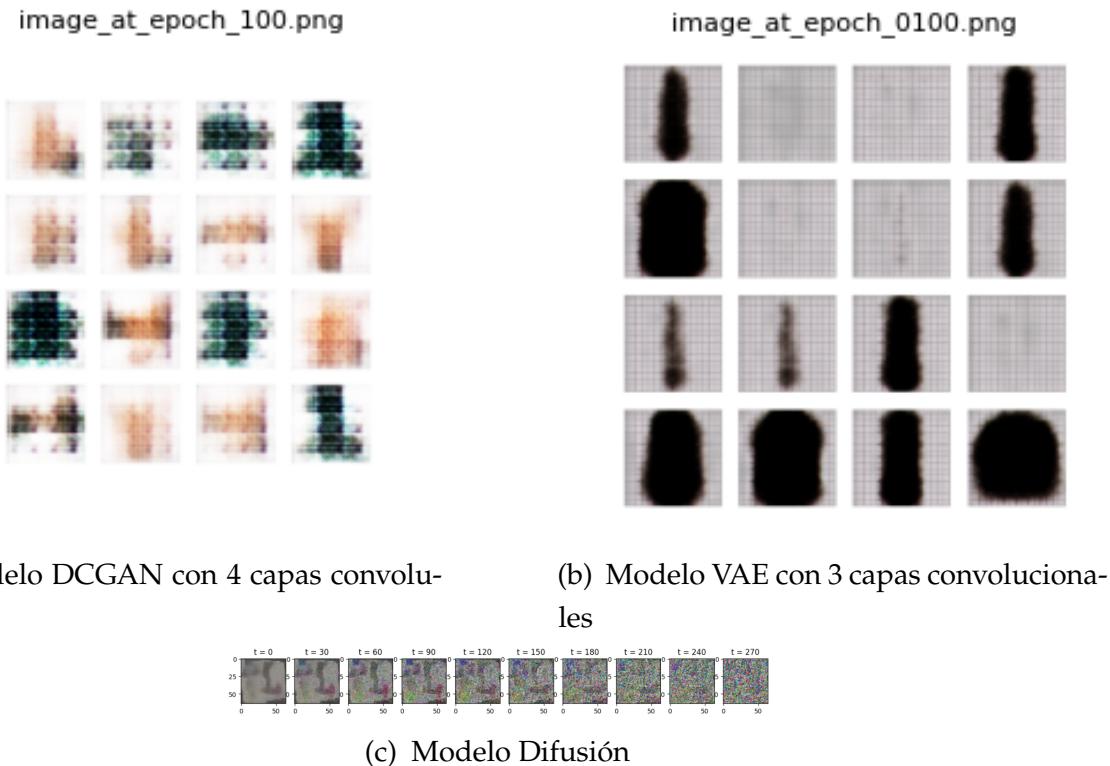
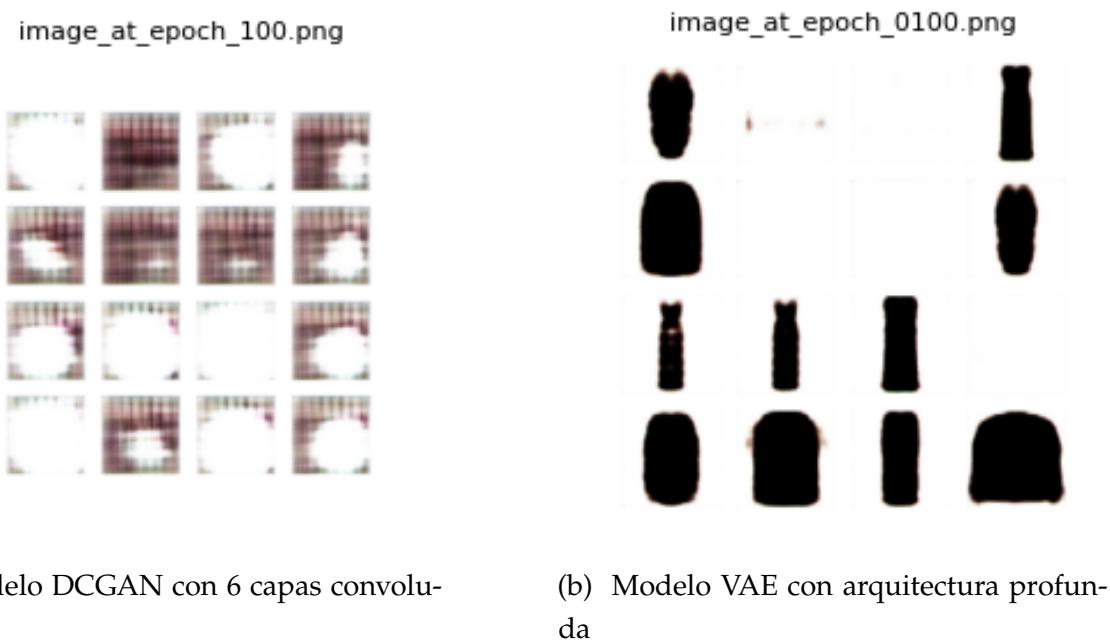


Figura 4.13. Comparativa de resultados entre tres modelos entrenados durante 100 épocas, con una arquitectura simple y empleando data augmentation

En la figura 4.14 vemos las imágenes generadas en la última *epoch* de los modelos DCGAN y VAE con más capas convolucionales que sus anteriores versiones y empleando *data augmentation*.

Como puede apreciarse, el modelo que consigue una mayor fidelidad visual y coherencia estructural es el **VAE** con una estructura más compleja, seguido por el su versión más simple. El resto de arquitecturas muestran limitaciones notables en cuanto a calidad visual, variedad y estructura de las muestras generadas.



(a) Modelo DCGAN con 6 capas convolucionales

(b) Modelo VAE con arquitectura profunda

Figura 4.14. Comparativa de resultados entre dos modelos entrenados durante 100 épocas, con una arquitectura convolucional más profunda y empleando data augmentation

4.3. Discusión de los resultados

Tras analizar los resultados, es posible identificar ciertas tendencias y limitaciones en el rendimiento de los modelos. Dada la propia naturaleza del *dataset* original, el cual carecía de variedad tanto en el tipo de prendas como en su distribución y considerando que el número de imágenes también era escaso; emplear técnicas de *data augmentation* resultó ser una estrategia clave. En el primer modelo, simplemente incorporando esta técnica de preprocesamiento y sin necesidad de modificar la arquitectura del modelo, los resultados mejoraron significativamente.

Con respecto a las distintas aproximaciones empleadas, el autoencoder variacional es el que ha producido las imágenes que mejor cumplían nuestro objetivo principal: generar nuevas imágenes de prendas de ropa. Sin embargo, las **Deep Convolutional Generative Adversarial Network** demostraron mayor capacidad para captar patrones cromáticos y texturas predominantes. Por último, el modelo de difusión ofrece el equilibrio perfecto entre fidelidad visual y variabilidad en la generación, siendo capaz tanto de mantener contornos definidos como de reproducir correctamente las paletas de color. Esta situación en la que cada modelo destaca en una tarea concreta nos lleva a desarrollar nuevos enfoques híbridos que combinan varias de estas técnicas con el fin de conseguir la má-

xima verosimilitud en las prendas.

Por último, los recursos computacionales de los que se disponía eran limitados, lo que nos motivó a experimentar con arquitecturas más ligeras y evitar saturaciones en el entorno de desarrollo. No obstante, la realización de este trabajo ha sentado las bases para comprender el proceso de implementación de modelos generativos, que alcanzarían resultados más notables si se diesen en una infraestructura más robusta, con más recursos y con un conjunto de datos más completo.

5.

Proyectos Futuros

Este trabajo se ha limitado a analizar y comparar el rendimiento y la capacidad generadora de tres enfoques utilizando el dataset creado mediante *web scrapping*. No obstante, las técnicas generativas abren la puerta a un innumerables proyectos nuevos, aplicaciones innovadoras y líneas de investigación.

En primer lugar, el limitado dataset motiva a replicar nuestro trabajo con un conjunto de datos más grande, con mayor resolución y más enfocado en el contexto de los desfiles de moda. La siguiente propuesta lógica es volver a reproducir el experimento con otros modelos como StyleGAN [8], [Visual Transformer \(ViT\)](#) [31] o combinaciones de varios, por ejemplo, [VAE + GAN](#). Una técnica muy empleada es el *fine-tuning* que consiste en utilizar un modelo grande previamente entrenado con grandes volúmenes de datos para una tarea general y entrenarlo con otros datos para adecuarlo a una tarea concreta. De esta forma, se ahorra mucho tiempo en el entrenamiento del modelo y se aprovechan los conocimientos previos aprendidos por el modelo base, lo que permite obtener buenos resultados incluso con conjuntos de datos más pequeños o específicos. Ejemplos de posibles modelos para aplicar *fine-tuning* son LoRa [32] y DreamBooth [33]. Finalmente, se puede añadir descripciones textuales para conseguir enfoques condicionados al texto de entrada y generar imágenes que cumplan unas características. Para tratar dichas descripciones textuales lo más común es utilizar CLIP [34]. Los modelos anteriormente mencionados permiten la manipulación de texto, pero también existen Stable Diffusion [22] o Kandisky 3 [35].

Una línea de investigación alternativa podría centrarse en tareas más específicas. Por ejemplo, generar prendas emulando a un diseñador concreto. De la misma forma que se ha propuesto introducir descripciones textuales, se podría pasar una imagen y que el modelo generara una colección de moda acorde a la estética o estilo de la imagen. También, existen técnicas de transferencia de estilo como Neural Style Transfer [36] se podrían emplear para tareas de transferencia de estilo, tejidos o diseñadores y Cycle GAN [9] que permite realizar traducción entre dominios sin necesidad de pares de imágenes alineadas. Esta técnica se podría aplicar en la conversión de prendas entre culturas, épocas históricas o estaciones, por ejemplo, transformar un vestido de invierno en su equivalente de verano. Alternativamente, otra futura línea sería la aplicación de

técnicas generativas para crear los patrones de confección de una prenda basada en unas características técnicas como las dimensiones corporales, asimetrías en la prenda o detalles textiles.

6.

Impacto

6.1. Impacto medioambiental

Este proyecto se centra en el diseño en la industria textil, una de las más contaminantes del mundo. La confección de prendas de ropa requiere una enorme cantidad de recursos naturales (agua, energía, materiales sintéticos) y genera una gran cantidad de residuos y emisiones de CO₂. Se estima que la industria de la moda es responsable de aproximadamente el 10 % de las emisiones globales de gases de efecto invernadero y del 20 % de las aguas residuales industriales [37].

En este sentido, el uso de la IA podría ayudar a mitigar el impacto medioambiental de varias formas: en primer lugar, la tecnología ofrece la ventaja de diseñar y simular físicamente los resultados sin necesidad de recurrir a textiles, materias primas y mano de obra. De esta forma se puede simular todo el proceso en entornos digitales eliminando así fases intermedias y desperdicio de recursos. Además, la fase de esbozar y crear los patrones resultaría superflua y, de nuevo, ahorrar en los materiales para llevar a cabo esas tareas.

Ahora bien, aunque el uso de la IA ayuda a reducir el consumo y gasto de recursos, se produciría un aumento significativo en la demanda de dispositivos electrónicos, servidores y centros de datos. Esto no solo implicaría un mayor consumo energético, sino también un incremento en la extracción de minerales, la generación de residuos electrónicos y la presión sobre los ecosistemas naturales. Además, si esta metodología se llegara a generalizar, aumentaría en gran medida el número de elementos electrónicos necesarios lo que provocaría un mayor consumo para conseguir la infraestructura necesaria.

En resumen, la IA generativa puede ser una herramienta clave para avanzar hacia una moda más sostenible, reduciendo los recursos y producción física. Sin embargo, lo que se gana por una parte se pierde por otra ya que el proceso de entrenamiento requiere el aumento en el uso de recursos tecnológicos podría llegar a ser incluso mayor que el actual.

6.2. Impacto social

A pesar de las ventajas que ofrece el uso de técnicas de IA, mientras una parte de la población lo arropa y lo recibe con los brazos abiertos otra gran parte de la población se muestra escéptica debido a las cuestiones éticas y filosóficas que plantean.

El mayor debate gira en torno a la autoría de estos diseños. ¿Quién debería ser considerado como autor del diseño: la persona que ha desarrollado la arquitectura del modelo o el propio modelo? Las diferentes respuestas a esta pregunta, a su vez, generan más cuestiones: por un lado, en caso de la persona que ha desarrollado el modelo sea considerada la autora, ¿hasta qué punto es responsable? Porque esta persona no se ajusta al concepto legal de autoría de una obra artística y ni experimenta todo lo que conlleva dicho título. Por otro lado, si se sugiere que el modelo informático es el verdadero autor, surge una paradoja legal: las inteligencias artificiales no pueden ostentar derechos de autor, ya que no tienen personalidad jurídica.

El problema de la autoría está estrechamente relacionado con el de la propiedad intelectual. Legalmente, solo una persona física puede ser el titular de la propiedad intelectual. Si la obra ha sido generada por una IA, ¿quién tomaría esa titularidad? O, por el contrario, ¿se debería permitir que esas creaciones no tuvieran derechos legales?

Además, el uso de modelos entrenados con datos visuales sin licencia plantea otro problema ético: ¿puede una IA inspirarse libremente en obras existentes sin infringir los derechos de los artistas originales? En muchos casos, estas obras se generan a partir de millones de imágenes recopiladas sin consentimiento explícito, lo cual ha generado demandas y protestas por parte de comunidades creativas.

Todos estos problemas han dado lugar a otras disyuntivas como la de si se puede considerar arte una creación hecha por un ordenador. Hay personas que piensan que todo resultado de un proceso creativo es arte indistintamente del mecanismo mediante el cual se ha generado, otros mantienen una postura más conservadora y afirman que el arte involucra la expresividad emocional y que los resultados generados por IA son imitativos y sin alma.

Por otro lado, en un aspecto más social y no tan filosófico, el uso de estas técnicas ha supuesto una democratización en el diseño de moda ya que muchos jóvenes artistas con bajos recursos tienen la oportunidad de compartir sus ideas simplemente con un ordenador, sin necesidad de poseer todo el dinero necesario para la confección de las

prendas. Igualmente, todo el mundo puede ser capaz de crear y generar sus propios diseños independientemente de su capacidad creativa. El lado negativo, una preocupación que ha estado muy presente recientemente es si la tecnología será capaz de sustituir los puestos de trabajo que hasta ahora han sido desempeñados por los humanos, y en particular, si la tecnología y los nuevos métodos generativos sustituirán a los artistas.

El uso de la tecnología en el arte ha conseguido democratizarlo y hacerlo accesible a todo el mundo, pero plantea graves debates éticos y filosóficos.

7.

Conclusiones

Este trabajo ha explorado el uso de técnicas de [inteligencia artificial](#) generativa en el diseño de prendas de ropa de diseñadores de moda de lujo mediante la implementación de tres de estas técnicas: [Deep Convolutional Generative Adversarial Network](#) que usa redes convolucionales profundas para capturar mejor las características visuales, [Variational Autoencoders](#) que trata de comprimir una imagen y reconstruirla posteriormente y modelos de difusión que aprenden a generar muestras nuevas a partir de un proceso de *denoising* aplicado a las muestras previamente difuminadas con ruido gaussiano. Cada una de estas técnicas presentaba unas ventajas y unos inconvenientes. La [DCGAN](#) no conseguía producir siluetas distinguibles, pero funcionaba bien captando patrones de color. Los [VAE](#) fueron los que produjeron resultados más concretos y siluetas identificables, pero no era capaz de reconstruir una imagen con paletas de color. Por último, el modelo de difusión proporcionó el mayor equilibrio entre generar formas diversas y captar los detalles visuales.

Para entrenar los modelos, se quiso utilizar un conjunto de datos con imágenes de prendas de ropa de las diferentes casas de moda de lujo. Por ello, se decidió crear el conjunto de datos desde cero utilizando *web scraping*. Los datos resultantes fueron escasos, menos de 2000 imágenes, y mostraban la imagen de los productos sobre fondo blanco. Esta escasez supuso un handicap a la hora de obtener resultados realistas y se afrontó aplicando técnicas de *data augmentation*. Con eso comprendimos la importancia que tiene la estructura y naturaleza de los datos.

En conclusión, aunque no se hayan obtenido imágenes visualmente coherentes, este proyecto ha servido para demostrar el potencial creativo que poseen estas técnicas y sentar las bases para futuras líneas de investigación centradas en mejorar la fidelidad de las imágenes generadas y personalizar la generación de moda. Posibles futuros proyectos incluyen realizar *fine-tuning* de modelos grandes ya preentrenados con conjuntos de datos muy voluminosos, aplicar otras arquitecturas más robustas o híbridas combinando varias de las técnicas desarrolladas, por ejemplo, [VAE](#) seguido de una [GAN](#). Por último, se podría ampliar el alcance de este proyecto y utilizar modelos que traten con descripciones textuales como CLIP para generar imágenes que cumplan ciertas características visuales.

A.

Repositorio del proyecto

Con el objetivo de facilitar el acceso al código fuente, imágenes generadas, arquitecturas y documentación técnica adicional, este trabajo ha sido subido a un repositorio público en GitHub. A través de este enlace, cualquier persona interesada podrá consultar los scripts utilizados, reproducir los resultados y profundizar en las técnicas empleadas:

[TFG_Patricia_Alcalde](#)

Este repositorio se encuentra bajo una licencia Creative Commons (CC BY-NC-SA 4.0), en línea con la licencia de la memoria de este TFG.

Referencias

- [1] K. Hornik, M. Stinchcombe y H. White, «Multilayer feedforward networks are universal approximators,» *Neural Netw.*, vol. 2, n.º 5, págs. 359-366, jul. de 1989.
- [2] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning Representations by Back-Propagating Errors,» *Nature*, vol. 323, n.º 6088, págs. 533-536, 1986. doi: [10.1038/323533a0](https://doi.org/10.1038/323533a0).
- [3] J. Kiefer y J. Wolfowitz, «Stochastic Estimation of the Maximum of a Regression Function,» *The Annals of Mathematical Statistics*, vol. 23, n.º 3, págs. 462-466, 1952. doi: [10.1214/aoms/1177729392](https://doi.org/10.1214/aoms/1177729392).
- [4] Y. LeCun, B. Boser, J. S. Denker et al., «Backpropagation Applied to Handwritten Zip Code Recognition,» *Neural Computation*, vol. 1, n.º 4, págs. 541-551, 1989. doi: [10.1162/neco.1989.1.4.541](https://doi.org/10.1162/neco.1989.1.4.541).
- [5] B. J. Frey, «Graphical models for machine learning and digital communication,» en *The handbook of brain theory and neural networks*, M. A. Arbib, ed., MIT Press, 2001, págs. 486-490.
- [6] D. P. Kingma y M. Welling, «Auto-Encoding Variational Bayes,» *arXiv preprint arXiv:1312.6114*, 2013.
- [7] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza et al., *Generative Adversarial Networks*, 2014. dirección: <https://arxiv.org/abs/1406.2661>.
- [8] T. Karras, S. Laine y T. Aila, «Analyzing and Improving the Image Quality of StyleGAN,» en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, págs. 8110-8119. dirección: <https://arxiv.org/abs/1912.04958>.
- [9] J.-Y. Zhu, T. Park, P. Isola y A. A. Efros, *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*, 2020. dirección: <https://arxiv.org/abs/1703.10593>.
- [10] M. Arjovsky, S. Chintala y L. Bottou, *Wasserstein GAN*, 2017. arXiv: [1701.07875 \[stat.ML\]](https://arxiv.org/abs/1701.07875). dirección: <https://arxiv.org/abs/1701.07875>.

- [11] A. Radford, L. Metz y S. Chintala, «Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,» *arXiv preprint arXiv:1511.06434*, 2015.
- [12] J. T. Springenberg, A. Dosovitskiy, T. Brox y M. Riedmiller, «Striving for Simplicity: The All Convolutional Net,» *arXiv preprint arXiv:1412.6806*, 2014.
- [13] S. Ioffe y C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,» *arXiv preprint arXiv:1502.03167*, 2015.
- [14] V. Nair y G. E. Hinton, «Rectified Linear Units Improve Restricted Boltzmann Machines,» en *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010, págs. 807-814.
- [15] B. Xu, N. Wang, T. Chen y M. Li, «Empirical Evaluation of Rectified Activations in Convolutional Network,» *arXiv preprint arXiv:1505.00853*, 2015.
- [16] A. L. Maas, A. Y. Hannun y A. Y. Ng, «Rectifier Nonlinearities Improve Neural Network Acoustic Models,» en *Proc. ICML*, vol. 30, 2013.
- [17] C. E. Shannon, «A Mathematical Theory of Communication,» *Bell System Technical Journal*, vol. 27, n.º 3, págs. 379-423, 1948.
- [18] D. P. Kingma y J. Ba, *Adam: A Method for Stochastic Optimization*, 2017. arXiv: [1412.6980 \[cs.LG\]](https://arxiv.org/abs/1412.6980). dirección: <https://arxiv.org/abs/1412.6980>.
- [19] D. Bank, N. Koenigstein y R. Giryes, *Autoencoders*, 2021. arXiv: [2003 . 05991 \[cs.LG\]](https://arxiv.org/abs/2003.05991). dirección: <https://arxiv.org/abs/2003.05991>.
- [20] P. Baldi, «Autoencoders, unsupervised learning and deep architectures,» en *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, 2011, págs. 37-50.
- [21] J. Ho, A. Jain y P. Abbeel, «Denoising Diffusion Probabilistic Models,» *arXiv preprint arXiv:2006.11239*, 2020.
- [22] R. Rombach, A. Blattmann, D. Lorenz, P. Esser y B. Ommer, «High-Resolution Image Synthesis with Latent Diffusion Models,» *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. dirección: <https://arxiv.org/abs/2112.10752>.
- [23] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan y S. Ganguli, «Deep Unsupervised Learning using Nonequilibrium Thermodynamics,» *arXiv preprint arXiv:1503.03585*, 2015.
- [24] M. Welling e Y. W. Teh, «Bayesian Learning via Stochastic Gradient Langevin Dynamics,» en *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 2011, págs. 681-688.

- [25] A. Ramesh, M. Pavlov, G. Goh et al., «Zero-Shot Text-to-Image Generation,» en *Proceedings of the 38th International Conference on Machine Learning (ICML)*, M. Meila y T. Zhang, eds., 2021.
- [26] P. Esser, R. Rombach y B. Ommer, «Taming Transformers for High-Resolution Image Synthesis,» en *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [27] J. Yu, X. Li, J. Y. Koh et al., *Vector-quantized Image Modeling with Improved VQGAN*, 2022. dirección: <https://arxiv.org/abs/2110.04627>.
- [28] J. Yi, T. Nasukawa, R. Bunescu y W. Niblack, «Sentiment Analyzer: Extracting Sentiments About a Given Topic Using Natural Language Processing Techniques,» en *Proceedings of the Third IEEE International Conference on Data Mining (ICDM 2003)*, IEEE, Melbourne, FL, USA, 2003, págs. 427-434.
- [29] Q. Zhang, V. S.-J. Huang, B. Wang et al., «Document Parsing Unveiled: Techniques, Challenges, and Prospects for Structured Information Extraction,» *arXiv preprint arXiv:2410.21169*, 2024, Survey.
- [30] O. Ronneberger, P. Fischer y T. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, 2015. dirección: <https://arxiv.org/abs/1505.04597>.
- [31] A. Dosovitskiy, L. Beyer, A. Kolesnikov et al., «An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,» en *International Conference on Learning Representations (ICLR)*, 2021. dirección: <https://arxiv.org/abs/2010.11929>.
- [32] E. J. Hu, Y. Shen, P. Wallis et al., «LoRA: Low-Rank Adaptation of Large Language Models,» *arXiv preprint arXiv:2106.09685*, 2021. dirección: <https://arxiv.org/abs/2106.09685>.
- [33] N. Ruiz, Y. Li, V. Jampani, Y. Pritch, M. Rubinstein y K. Aberman, «DreamBooth: Fine Tuning Text-to-Image Diffusion Models for Subject-Driven Generation,» *arXiv preprint arXiv:2208.12242*, 2022. dirección: <https://arxiv.org/abs/2208.12242>.
- [34] A. Radford, J. W. Kim, C. Hallacy et al., «Learning Transferable Visual Models From Natural Language Supervision,» en *Proceedings of the 38th International Conference on Machine Learning (ICML)*, 2021. dirección: <https://arxiv.org/abs/2103.00020>.

- [35] V. Arkhipkin, V. Vasilev, A. Filatov et al., «Kandinsky 3: Text-to-Image Synthesis for Multifunctional Generative Framework,» en *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing (EMNLP Demo)*, nov. de 2024, págs. 475-485. dirección: <https://arxiv.org/abs/2410.21061>.
- [36] L. A. Gatys, A. S. Ecker y M. Bethge, *A Neural Algorithm of Artistic Style*, 2015. dirección: <https://arxiv.org/abs/1508.06576>.
- [37] Ellen MacArthur Foundation, «A New Textiles Economy: Redesigning Fashion's Future,» Ellen MacArthur Foundation, 2017, Accessed: 2025-07-10. dirección: <https://ellenmacarthurfoundation.org/a-new-textiles-economy>.
- [38] P. Esser, R. Rombach y B. Ommer, *Taming Transformers for High-Resolution Image Synthesis*, 2021. arXiv: [2012.09841 \[cs.CV\]](https://arxiv.org/abs/2012.09841). dirección: <https://arxiv.org/abs/2012.09841>.

Índice de términos

Glosario

Computer Aided Design Diseño Asistido por Computadora, es una tecnología que utiliza software para ayudar en el proceso de diseño, permitiendo a los usuarios crear, modificar, analizar y documentar diseños en 2D y 3D.. [1](#)

CycleGAN algoritmo de aprendizaje profundo que permite la traducción de imágenes de un dominio a otro sin necesidad de pares de imágenes de entrenamiento.. [15](#)

DALL·E sistema de IA que puede crear arte e imágenes realistas a partir de una descripción en lenguaje natural.. [30](#)

Denoising Diffusion Probabilistic Models los Modelos Probabilísticos de Difusión con Denoisificación son un tipo de modelo generativo que se utiliza para generar datos, especialmente imágenes, mediante un proceso de inversión de un ruido. .
[28](#)

Internet of Things Red de objetos físicos conectados a Internet a través de software y sensores . [31](#)

Stable Diffusion modelo de aprendizaje automático desarrollado por Runway y LMU Múnich para generar imágenes digitales de alta calidad a partir de descripciones en lenguaje natural.. [28](#)

StyleGAN tipo de red generativa adversarial desarrollada por Nvidia que se especializa en generar imágenes realistas de rostros humanos, entre otros tipos de imágenes.. [15](#)

Telar de Jacquard máquina de tejer mecánica inventada por Joseph Marie Jacquard que permite la creación de patrones complejos y detallados en la tela mediante el uso de tarjetas perforadas para controlar los hilos individuales.. [1](#)

VQ-GAN combina los beneficios de los VQ-VAE con el poder adversarial de los GANs.

Fue propuesto por Esser et al. en 2021 [38]y mejora la calidad perceptiva de las imágenes generadas usando una pérdida adversarial durante el entrenamiento..

31

Siglas

AE autoencoders. 24, 25

AE autoencoder. 24–26

CNN Convolutional Neural Network. 11, 14, 19

DCGAN Deep Convolutional Generative Adversarial Network. 15, 19, 21, 42, 50, 57, 59, 66

ELBO Evidence Lower Bound. 27, 28, 44, 54, 56

GAN Generative Adversarial Network. 1–3, 15–19, 37, 61, 66

IA inteligencia artificial. 1–3, 5, 31, 63, 64, 66

KL Kullback-Leibler. 14, 26, 27

MSE Mean Squared Error. 25, 30, 49

PCA Análisis de Componentes Principales. 25

VAE Variational Autoencoders. 1, 3, 25, 26, 28, 30, 54, 56, 61, 66

ViT Visual Transformer. 61

