UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Sistemas Informáticos y Computación

Estudio y comparativa de diferentes modelos de difusión para generación condicional de caras

Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

AUTOR/A: Juan González, Jorge

Tutor/a: Paredes Palacios, Roberto

CURSO ACADÉMICO: 2022/2023

Índice general

Capítulo 1

Introducción

1.1. Problema

Los modelos de difusión son una clase emergente de modelos generativos que han demostrado ser altamente efectivos para generar imágenes realistas. Estos modelos han demostrado ser útiles en una variedad de aplicaciones, incluyendo la generación de imágenes realistas, edición de imágenes y superresolución, entre otras. También han demostrado ser eficaces en la modelización de otros tipos de datos como audio o datos de texto.

Uno de los desafíos más destacados en el campo de la generación de imágenes es la creación de caras humanas realistas. Conseguir un modelo de difusión capaz de generar caras realistas es de vital importancia por sus implicaciones prácticas. Entre otras aplicaciones, puede usarse en seguridad para mejorar sistemas de reconocimiento facial, o en vigilancia para recuperar fotos borrosas o incompletas. Además, las caras humanas tienen una estructura intrincada y compleja que las hace un desafío particular para los modelos generativos. Son extremadamente ricas en detalles y variaciones sutiles que, si se modelan incorrectamente, pueden resultar en caras irreales. Lograr la generación de caras realistas es una prueba de fuego para los modelos generativos, y un modelo de difusión capaz de lograr esto demostraría su capacidad para modelar con éxito datos altamente estructurados y complejos.

Otra consideración importante es la capacidad del modelo para generar datos que son consistentes con unas condiciones o restricciones dadas. Esto es esencial en muchas aplicaciones prácticas de los modelos generativos, donde no solo queremos generar datos que sean realistas, sino también que cumplan con ciertos criterios. Por ejemplo, en la generación de caras, podemos querer generar caras que no solo sean realistas, sino también que muestren a una persona con unos rasgos concretos o que generen variaciones a partir de la foto de una persona.

1.2. Alcance

El objetivo principal de este trabajo será investigar y comparar diferentes modelos de difusión para determinar cuál resulta más fiable a la hora de generar caras que cumplan una serie de atributos. Analizaremos la calidad de las imágenes generadas y la fidelidad con la que estas imágenes se ajustan a la condición.

Como se explica más adelante, entre estas métricas se encuentra CLIP Score que es la métrica más utilizada para medir la calidad del condicionamiento en modelos generativos de imágenes condicionados por texto (modelos text-to-image). CLIP Score utiliza el modelo CLIP, encargado de estimar cuánto se corresponde un texto con una imagen, por lo que está sujeta al sesgo que pueda tener este modelo. Para reforzar los resultados de la evaluación entrenaremos un modelo que estime la condición usada a partir de la imagen generada con el objetivo de usar este modelo como métrica complementaria a CLIP Score.

Finalmente, dado que el entrenamiento de modelos de difusión aún requiere una cantidad considerable de recursos, en este trabajo estudiaremos y aplicaremos técnicas que permitan entrenar estos modelos de forma más eficiente.

1.3. Limitaciones

Existen modelos de difusión con más parámetros que los tratados en este trabajo. Estos modelos nos permitirían generar imágenes de más calidad que las presentadas en este trabajo pero entrenar estos modelos queda fuera de nuestro alcance por limitaciones de recursos.

Además, evaluar la calidad de una imagen o la correspondencia entre una imagen y una condición son tareas inherentemente subjetivas. Por ello, las métricas que se usan implican hacer inferencia a un modelo entrenado para simular el criterio humano por lo que estas métricas y sus resultados estarán sujetos a los sesgos que puedan tener esos modelos.

1.4. Estructura

En este trabajo haremos un análisis teórico de los principales componentes y conceptos relacionados con los modelos de difusión. Empezaremos con una exploración general de lo que son estos modelos y cómo pueden generar imágenes. Después, centraremos nuestro análisis en una variante específica de estos modelos, los Denoising Diffusion Probabilistic Models (DDPM). Seguiremos con un examen de las arquitecturas de redes neuronales más utilizadas en la implementación de estos modelos: U-Net y Transformer. Continuaremos con la descripción de 'Stable Diffusion', un modelo de difusión guiado por texto, y examinaremos LoRA, un enfoque eficiente para el entrenamiento de modelos

pre-entrenados. Profundizaremos en los los mecanismos que existen para condicionar la inferencia de un modelo de difusión y revisaremos los datasets de imágenes faciales más significativos.

Una vez expuesto el Estado del Arte, detallaremos la implementación de los dos modelos que hemos evaluado y explicaremos las métricas empleadas para dicha evaluación, así como los datos con los que fueron entrenados. Seguidamente, proporcionaremos una descripción de cómo se desarrollaron los experimentos y qué resultados hemos obtenido, discutiendo lo que podemos extraer de ellos.

Finalizaremos este trabajo con unas conclusiones en las que destacaremos las aportaciones y los resultados más relevantes.

Capítulo 2

Estado del Arte

2.1. DDPM

Los modelos de difusión son modelos generativos que definen una serie de pasos de difusión en los que se añade iterativamente ruido aleatorio a los datos. Después, aprenden a invertir el proceso de difusión para generar nuevas muestras a partir del ruido. Se han propuesto varios modelos basados en esta idea pero los *Denoising Diffusion Probabilistic Models* (DDPM), propuestos por Ho et al. en 2020 [15], demostraron ser capaces de generar imágenes de gran calidad. En este trabajo, utilizaremos el término 'modelo de difusión' para referirnos a DDPM y a sus variantes.

Los modelos de difusión sintetizan muestras nuevas a través de un proceso iterativo de eliminación de ruido y constituyen actualmente el Estado del Arte en síntesis de imágenes. Recientemente han atraído mucha atención después de que empresas como OpenAI y Google consiguieran generar imágenes con un realismo sin precedentes usando este tipo de modelos. Algunos de estos modelos de difusión son GLIDE, DALLE-2, Imagen, Stable Diffusion y DeepFloyd IF.

Existen otros modelos generativos capaces de sintetizar imágenes como pueden ser los Variational Autoencoders (VAE) o las Generative Adversarial Networks (GAN).

Un Variational Autoencoder, propuesto por Kingma et al. en 2013 [23], es un tipo de modelo que aprende a codificar datos de entrada en un espacio de representación de menor dimensión (conocido como espacio latente), siendo capaz de generar nuevas muestras a partir de puntos en este espacio latente. La característica más notable de los VAE es que aprende a mapear los datos de entrada sobre una distribución normal. Esto le permite generar muestras plausibles a partir de puntos del espacio latente que no se corresponden necesariamente con la representación latente de una muestra de entrenamiento. Sin embargo, los

VAE suelen generar muestras más borrosas en comparación con otros modelos generativos, como las redes GAN.

Las redes GAN, propuestas por Goodfellow et al. en 2014 [9], constan de dos modelos que se entrenan de forma conjunta en un juego de suma cero. Un modelo generador se entrena para generar muestras que lleguen a ser interpretadas como reales por un modelo discriminador. A su vez, el modelo discriminador se entrena para minimizar sus fallos de detección frente al modelo generador. Este tipo de redes llegaron a dominar el campo de la síntesis de imágenes porque permiten generar muestras de muy alta calidad. Sin embargo, el entrenamiento de estas redes es muy inestable y no llegan a modelar correctamente la diversidad de las muestras. En 2021, Dhariwal y Nichol argumentaron que los modelos de difusión podían superar a las redes GAN en la tarea de síntesis de imágenes en términos de calidad y diversidad [6].

Un modelo de difusión permite generar muestras de alta calidad como las GAN pero con un entrenamiento mucho más estable y cubriendo mejor la diversidad de las muestras. Sin embargo, el principal inconveniente de estos modelos es que su proceso de muestreo requiere mucho más tiempo que el de un VAE o una GAN al ser necesario hacer inferencia a la red en cada paso del proceso de muestreo.

Es posible convertir una distribución compleja en una distribución Gaussiana isotrópica perturbando leve pero iterativamente las muestras añadiéndoles ruido, tal y como se muestra en la Figura 2.1. Si la distribución de las perturbaciones aplicadas en cada iteración son conocidas es posible transformar la distribución Gaussiana de vuelta a la distribución compleja. Este es el principio fundamental sobre el que se sustentan los modelos de difusión [31].

El proceso iterativo por el cual perturbamos una muestra hasta convertirla en una muestra de ruido Gaussiano se conoce como proceso de difusión. Existen distintas formas de llevar a cabo este proceso pero analizaremos y usaremos el método original, descrito por Ho et al. en 2020 [15].

La condición que deben cumplir esas perturbaciones, esas muestras de ruido, es que deben ser muestras de distribuciones Gaussianas donde la media sea la muestra del paso anterior $(t-1)$:

$$\mathbf{x}_t \sim \mathcal{N}(\mathbf{x}_{t-1}, \mathbf{I}) \iff \mathbf{x}_t = \mathbf{x}_{t-1} + \mathcal{N}(\mathbf{0}, \mathbf{I})$$

Después de T iteraciones o pasos, las muestras perturbadas se habrán convertido en ruido Gaussiano, es decir, seguirán una distribución Gaussiana isotrópica:

$$q(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$

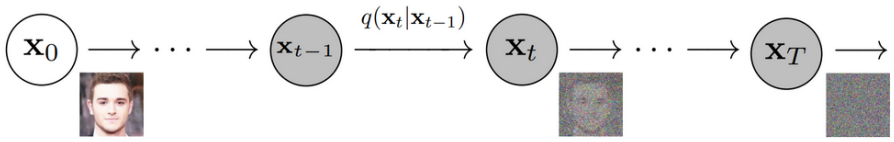


Figura 2.1: Forward diffusion. [15]

En la práctica, la varianza de cada distribución de perturbación $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ tiene un valor predefinido que depende de t. Estos valores, conocidos como plan de varianza o variance schedule y denotados como $\beta_{1:T}$, son realmente hiperparámetros que se eligen antes del entrenamiento y determinan cómo varía la varianza de la distribución $q(\mathbf{x}_t|\mathbf{x}_{t-1})$ conforme avanza el proceso de difusión.

Esta serie de valores puede ser constante o variar a lo largo de los T pasos. Una forma de variarlo es utilizar un plan que puede ser lineal, cuadrático, coseno, entre otros. Ho et al. [15] utilizaron un programa lineal en el que la varianza aumentaba de $\beta_1 = 10^{-4}$ a $\beta_T = 0,02$ en su propuesta de DDPM.

Si tenemos en cuenta $\beta_{1:T}$, la distribución de perturbación quedaría como se indica:

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

De acuerdo con lo anterior, para estimar el ruido que se ha añadido a la muestra $\mathbf{x}_0$ en el paso t es necesario obtener $\mathbf{x}_{t-1}$ lo que implicaría hacer el proceso de difusión hasta el paso $t-1$. Para agilizar el proceso podemos utilizar el truco de reparametrización [46]. Este truco nos permite conocer la distribución de ruido con la que perturbar $\mathbf{x}_0$ para llegar a $\mathbf{x}_t$ en un sólo paso tal y como se describe a continuación.

Siendo $\alpha_t = 1 - \beta_t$ y $\bar{\alpha}_t = \prod_{s=0}^{t} \alpha_s$, se puede demostrar que:

$$\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$$

$$\tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t$$

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t\mathbf{I})$$

Nótese que α_t y $\bar{\alpha}_t$ sólo dependen de β_t, por lo que pueden ser precalculados.

Cabe destacar que, dado que las perturbaciones realizadas en una iteración o paso de difusión sólo dependen del anterior, la serie de pasos de difusión constituye una cadena de Markov. Además, la distribución final a la que llegamos tras el proceso de difusión $q(\mathbf{x}_T)$ será más próxima a una distribución

Gaussiana isotrópica cuantos más pasos tenga el proceso de difusión, es decir, $q(\mathbf{x}_T) = \mathcal{N}(\mathbf{0}, \mathbf{I}) \iff T \to \infty$ [41] por lo que T será por lo general un valor alto.

El objetivo de estos modelos es aprender el proceso inverso, es decir, estimar la perturbación que se añadió en un paso t conociendo $\mathbf{x}_t$ tal y como se muestra en la Figura 2.2. Dicho de otro modo, entrenaremos la red para que genere muestras de ruido que cumplan con la distribución $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$:

$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t); \mathbf{\Sigma}_\theta(\mathbf{x}_t, t))$$

Ho et al. [15] probaron empíricamente que usar $\Sigma_\theta(\mathbf{x}_t) = \sigma^2\mathbf{I} = \beta_t\mathbf{I}$ da buenos resultados por lo que se puede simplificar la tarea y entrenar la red para predecir sólo la media $\mu_\theta(\mathbf{x}_t, t)$.

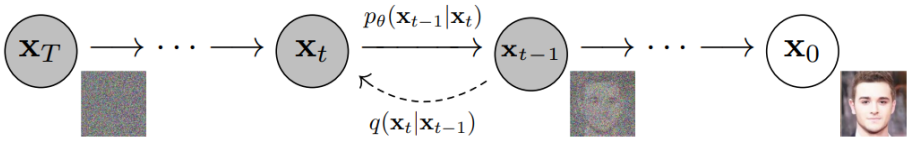


Figura 2.2: Backward diffusion. [15]

Ahora bien, para optimizar los parámetros de la red es necesario calcular una función de pérdida. El objetivo de un modelo generativo basado en verosimilitud es conseguir que las muestras generadas sigan la distribución de probabilidad de los datos $p(\mathbf{x})$. Idealmente, convendría usar log-verosimilitud negativa como función de pérdida:

$$L := -log\, p_\theta(\mathbf{x}_0)$$

Sin embargo, no podemos calcular $p_\theta(\mathbf{x}_0)$ directamente porque tendríamos que considerar todas las formas posibles en las que el proceso de difusión podría haber empezado en $\mathbf{x}_0$ y acabado en la distribución $p(\mathbf{x}_T)$. Dicho de otro modo, implicaría integrar sobre todas las trayectorias posibles del proceso de difusión, lo que resulta intratable en distribuciones complejas donde D es alto como ocurre cuando trabajamos con imágenes:

$$p_\theta(\mathbf{x}_0) = \int p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_t|\mathbf{x}_{t-1}) d\mathbf{x}_0 ... d\mathbf{x}_T$$

La cota inferior de la evidencia o cota inferior variacional (ELBO o VLB, por sus siglas en inglés) [21] es un método para convertir problemas de inferencia estadística originalmente intratables en problemas de optimización que se pueden aproximar con redes neuronales.

En lugar de minimizar una log-verosimilitud negativa desconocida, minimizaremos el valor esperado de otra expresión que sí podemos calcular y que

sabemos que siempre será mayor o igual que $\mathbb{E}[-log\, p_\theta(\mathbf{x}_0)]$. En el caso de los modelos de difusión [40], esta desigualdad se puede expresar como:

$$\mathbb{E}[-\log p_\theta(\mathbf{x}_0)] \leq \mathbb{E}_q \left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right] \tag{2.1}$$

$$= \mathbb{E}_q \left[-\log p(\mathbf{x}_T) - \sum_{t \geq 1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})} \right] =: L \tag{2.2}$$

Esta expresión constituye la función de pérdida que trataremos de minimizar y se puede reformular como:

$$\mathbb{E}_q \left[\underbrace{D_{KL}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{-log\, p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0} \right] \tag{2.3}$$

Podemos simplificar esta expresión aún más e ignorar los términos L_0 y L_T:

$$L := L_{t-1} := D_{KL}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))$$

Finalmente esta D_{KL}, siguiendo la derivación de los autores [15] y siendo $\sigma_t^2 = \beta_t$, se puede calcular como:

$$L := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \left[\frac{\beta_t^2}{2\sigma_t^2 \alpha_t (1-\bar{\alpha}_t)} ||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)||^2 \right]$$

En la práctica, se suele simplificar esta expresión al error cuadrático medio (MSE, por sus siglas en inglés) entre el ruido estimado por el modelo $\boldsymbol{\epsilon}_\theta \sim p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ y una muestra de ruido Gaussiano $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$:

$$L_{\text{simple}} := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \left[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)||^2 \right]$$

$$L_{\text{simple}} := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}} \left[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)||^2 \right]$$

Una vez somos capaces de estimar $\boldsymbol{\epsilon}_t \sim p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$, podemos tomar una muestra $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ e ir refinándola iterativamente, eliminando el ruido estimado en cada paso, hasta conseguir una muestra de una distribución que idealmente se aproximará mucho a la distribución de los datos de entrenamiento. A esto se le conoce como proceso de muestreo e involucra hacer inferencia al modelo muchas veces, razón por la cual estos modelos requieren más tiempo que otros modelos generativos para generar una muestra.

Hay diferentes formas de eliminar el ruido a partir de una imagen ruidosa y la estimación de nuestro modelo. En este trabajo utilizaremos el mismo método que Ho et al. utilizaron en su artículo sobre DDPM [15]. Este método de muestreo permite obtener $\mathbf{x}_{t-1}$ de la siguiente manera:

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

Podemos estimar el resultado del proceso de muestreo $\hat{\mathbf{x}}_0$ aprovechando la información parcial contenida en $\mathbf{x}_t$ de la siguiente forma:

$$\mathbf{x}_0 \approx \hat{\mathbf{x}}_0 = (\mathbf{x}_t - \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t))/\sqrt{\bar{\alpha}_t}$$

Por lo general, los modelos de difusión tienen la propiedad de funcionar con diferentes métodos de muestreo. Dicho de otro modo, es posible llevar a cabo el proceso de muestreo usando distintos métodos de muestreo para un mismo modelo pre-entrenado.

Dado que existen distintas formas de realizar el proceso de difusión y el proceso de muestreo, se considera una buena práctica en la implementación de estos modelos el hecho de crear una clase que contenga los métodos que se van a usar tanto para añadir ruido en cada paso del proceso de difusión, como para conseguir una imagen menos ruidosa en cada paso del proceso de muestreo. A esta clase se le conoce como planificador de ruido (noise scheduler).

En el planificador de un DDPM, el método de muestreo, aunque efectivo, requiere necesariamente realizar T pasos, es decir, hacer T inferencias. Recordemos que T es por lo general un número alto por lo que el proceso de muestreo propuesto por Ho et al. [15] resulta lento.

En 2021, Nichol et al. [28] demostraron que el uso de un programa de varianza coseno daba mejores resultados. La Figura 2.3 muestra la evolución del proceso de difusión utilizando los programas linear y coseno.

Figura 2.3: Programa linear (arriba) y coseno (abajo) respectivamente. [28]

Durante el entrenamiento, el modelo debe aprender distintas distribuciones $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ optimizando pesos compartidos. Optimizar los pesos en función de la loss obtenida para un paso puede perjudicar el desempeño del modelo en otro paso. Dicho de otro modo, existen gradientes conflictivos.

En marzo de 2023, Hang et al. [12] propusieron una estrategia de ponderación de la loss, Min-SNR-γ, con la que tratan cada paso t como una tarea individual, escalando la loss en función de su dificultad. Esta dificultad viene

determinada por el paso de denoising t para el que estemos calculando la loss: cuando $t \to 0$ la tarea de denoising es más fácil.

Los autores proponen y demuestran que escalar la loss por Min-SNR-γ permite al modelo converger más rápido tal y como se puede ver en la figura 2.4. Esta mejora también se observa para diferentes arquitecturas. En el caso de los modelos que predicen ruido, como son los que tratamos en este trabajo, esta estrategia propone escalar la loss por $min\{\gamma/SNR(t), 1\}$ donde $SNR(t) = \alpha_t^2/\sigma_t^2$ y γ es un hiperparámetro. Los autores recomiendan usar $\gamma = 5$.

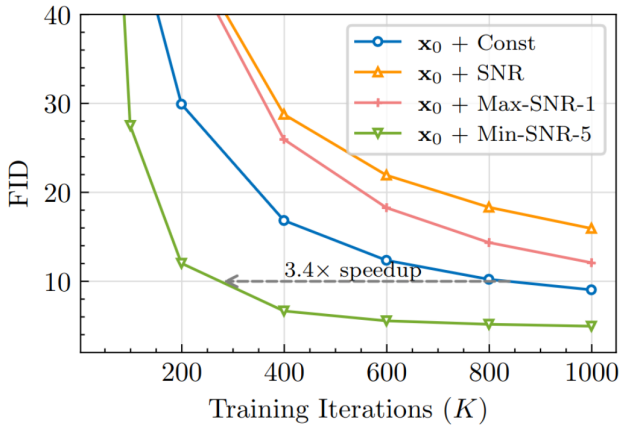


Figura 2.4: Evolución de la loss (ImageNet 64x64). [12]

2.2. Arquitectura U-Net

Un requisito que debe cumplir la red encargada de modelar $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ es que las dimensiones de la entrada y la salida sean idénticas. La arquitectura de red más utilizada con esta característica es la red U-Net, como la que se muestra en la Figura 2.5, propuesta originalmente por Olaf Ronneberger et al. en 2015 [35]. Esta conocida arquitectura ha sido objeto de numerosas modificaciones para mejorar su precisión.

Una U-Net es una arquitectura de red neuronal convolucional (CNN, por sus siglas en inglés). Se denomina U-Net porque tiene una arquitectura en forma de U, con una serie de niveles de contracción y expansión donde la salida de un nivel de contracción se concatena con la entrada de su correspondiente nivel de expansión.

La diferencia respecto a una arquitectura Codificador-Decodificador reside en que en una arquitectura U-Net las capas de expansión y contracción del

mismo tamaño están conectadas, lo que permite a la red recuperar información que puede haberse perdido en la codificación y que puede ser relevante en la decodificación. Estas conexiones se conocen como "skip connections".

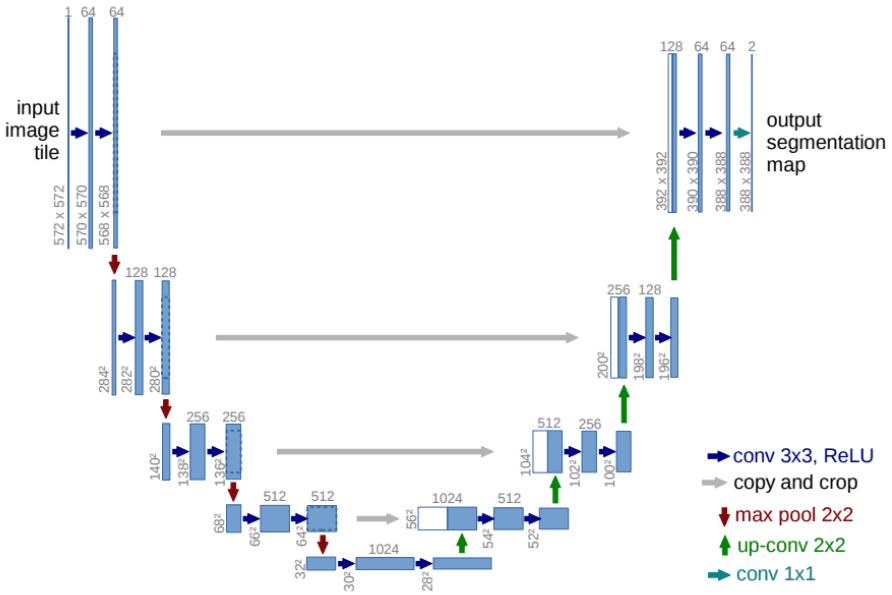


Figura 2.5: Arquitectura U-Net. [35]

Un concepto importante en redes convolucionales es el de conexión residual. Las CNN con muchas capas presentan el problema del descenso de gradiente que dificulta el aprendizaje de las capas más profundas. Esto puede resolverse tomando la entrada a un grupo de capas de la red y sumándola a la salida, tal y como se muestra en la Figura 2.6, en lo que se conoce como bloque residual. De esta forma el gradiente puede pasar por la conexión residual hacia capas más profundas sin desvanecerse.

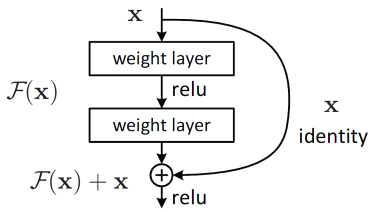


Figura 2.6: Bloque residual.

En 2019, Yang et al. propuso Res-UNet [49], una U-Net compuesta de bloques residuales que permiten hacer redes con más capas y, por ende, capaces de extraer más características. Debido a su mayor rendimiento y efectividad en la extracción de características, Res-UNet se utiliza ampliamente como modelo base para muchas arquitecturas profundas [47].

En 2020, Zhang et al. combinaron puertas de atención con una Res-UNet para la segmentación de tumores cerebrales y han demostrado ser más eficaces frente a la Res-UNet sin puertas de atención [51].

Una alternativa a la U-Net es la arquitectura Vision Transformer (ViT), que trata las imágenes como una secuencia de parches de igual tamaño, procesándolos con un Transformer estándar [7]. Aunque originalmente se diseñó para la clasificación de imágenes, adaptaciones recientes de ViT han demostrado ser eficaces también en generación de imágenes y, específicamente, en modelos de difusión como es el caso de DiT [32].

Actualmente, los modelos text-to-image que constituyen el Estado del Arte en generación de imágenes utilizan redes U-Net para modelar $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ por lo que son las que usaremos en este trabajo.

2.3. Arquitectura Transformer

Unos conceptos ampliamente utilizados dentro del área del procesamiento del lenguaje natural (NLP, por sus siglas en inglés) desde 2014 son los mecanismos de atención y los mapas de atención [1].

Un *mapa de atención* es un tensor con dimensión $N \times M$ donde N y M son el número de elementos de dos muestras de entrada. Cada elemento del mapa de atención tendrá un valor entre 0 y 1 que, después del entrenamiento, indicará cuán relacionado está cada elemento de N con cada elemento de M. Si las dos muestras de entrada son la misma, el mapa de atención estaría representando la relación entre los elementos de una misma muestra y en ese caso tendría dimensión $N \times N$.

Un *mecanismo de atención* es una serie de operaciones que producen un mapa de atención a partir de 2 proyecciones distintas de una primera entrada (K y V) y 1 proyección de una segunda entrada (Q). Cuando las dos entradas son iguales nos referimos a esta serie de operaciones como *self-attention*. Cuando K y V son proyecciones de una entrada pero Q es proyección de otra entrada diferente, nos referimos a esta serie de operaciones como *cross-attention*. Estas *proyecciones* son la salida de una o varias capas de redes neuronales (MLPs). Es esta correcta proyección la que se aprenderá durante el entrenamiento del modelo.

Si un mapa de atención pondera cada elemento de la salida de una red neuronal decimos que el mapa de atención actúa como *puerta de atención*.

Cuando entrenamos varios mecanismos de atención en paralelo podemos conseguir que el mapa de atención producido por cada mecanismo de atención represente una relación distinta. En estos casos, cada mecanismo de atención constituye una *cabeza* y al conjunto se le denomina capa de *atención de múltiples cabezas*.

Existen diferentes mecanismos de atención pero el más conocido es Scaled Dot-Product Attention, mostrado en la Figura 2.7, y que se calcula como:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d}}\right) \cdot V$$

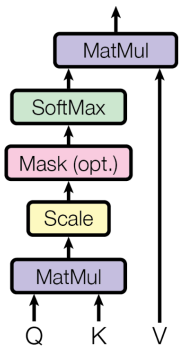


Figura 2.7: Scaled Dot-Product Attention. [43]

Las puertas de atención se pueden usar en redes convolucionales como se muestra en la Figura 2.8. Permiten a la red centrarse en regiones específicas de una imagen o mapa de características, mejorando así sus resultados sin incrementar de forma significativa el número de parámetros ni el tiempo de inferencia de la red. Su uso ha permitido mejorar resultados también en segmentación de tumores cerebrales [30] y otras tareas dentro del campo de visión por ordenador.

En 2017, Vaswani et al. presentaron la arquitectura Transformer [43]. Esta arquitectura de red neuronal ha revolucionado el campo del procesamiento del lenguaje natural, siendo la base de muchos modelos que constituyen hoy el Estado del Arte como BERT, GPT-4 y T5. La capacidad de los Transformers para generar representaciones ricas de los datos de entrada ha demostrado ser extremadamente valiosa en una amplia variedad de tareas.

Un Transformer se compone de un codificador y un decodificador que explotan el concepto de atención de múltiples cabezas. Tanto la serie de valores

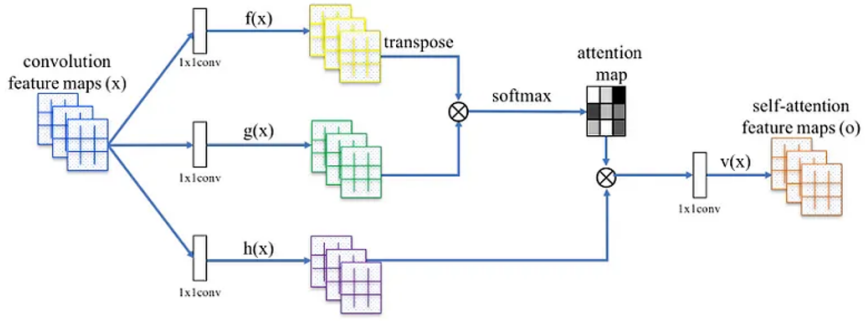


Figura 2.8: Puerta de atención de una CNN. [50]

de entrada como la serie de valores esperados a la salida se someten a una codificación posicional que permite estimar la serie completa de salida con una sola inferencia durante el entrenamiento. En inferencia, dado que no hay una serie de valores esperados, el Transformer actúa como un modelo autorregresivo, realizando una inferencia para cada token predicho hasta completar la serie. La Figura 2.9 muestra los componentes de un Transformer.

Una restricción de estos modelos es que necesitan como entrada una serie de t tokens de dimensión d. Para procesar imágenes usando Transformers se deben convertir estas imágenes a una serie de tokens, de forma similar a como un texto se debe tokenizar en tareas de NLP. Estos tokens se usan como entrada para el Transformer e idealmente, tras el entrenamiento, el modelo será capaz de reconocer patrones visuales y su relación espacial dentro de la imagen, algo para lo que tradicionalmente se han usado CNNs. Gracias a los mecanismos de atención, el modelo puede entender el contexto y las relaciones entre zonas de la imagen, de manera similar a como en un texto se comprende el contexto y las relaciones entre palabras.

Esta tokenización puede hacerse simplemente tomando cada píxel como un token y proyectarlo a d dimensiones. Esta aproximación, aunque puede funcionar para imágenes de baja resolución, puede ser ineficiente y poco práctico para imágenes de alta resolución debido a la cantidad de tokens que se necesitan procesar.

La alternativa más utilizada consiste en dividir la imagen de dimensión $w \times h \times c$ en parches de dimensión $p \times p \times c$, disponer en serie cada valor del parche y tratar cada parche como un token de dimensión p^2c. Convertir los parches a una dimensión destruye su información espacial. Para no perderla, se suma al vector resultante un embedding posicional que asigna un valor único a cada píxel en función de su posición en la imagen.

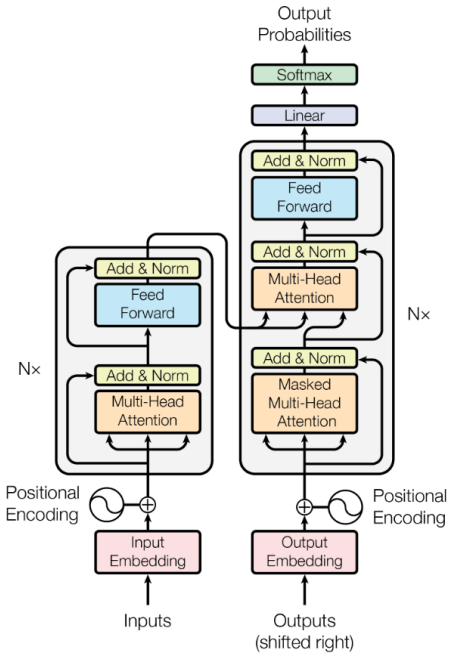


Figura 2.9: Arquitectura Transformer. [43]

Esta aproximación es la que siguen los Vision Transformers (ViT), propuestos por Dosovitskiy et al. en 2020 [7]. Actualmente es la forma más común de procesar imágenes con Transformers dado que es eficiente y hace factible el procesamiento de imágenes de alta resolución. La Figura 2.10 muestra cómo se tokeniza una imagen en un ViT.

2.4. Stable Diffusion

Los modelos de difusión como DDPM llevan a cabo los procesos de difusión y muestreo en el espacio del píxel, es decir, la muestra de ruido $\boldsymbol{\epsilon}$ aplicado a $\mathbf{x}_t$ en el paso t tendrá tantos píxeles como $\mathbf{x}_0$. Esto obliga al modelo a aprender detalles de alta frecuencia que son muchas veces imperceptibles, lo que se traduce en largos tiempos de entrenamiento y en la imposibilidad práctica de escalar estos modelos para imágenes de alta resolución.

Una posible solución a este problema de escalabilidad es usar modelos de superresolución en serie, tal y como propusieron Ho et al. [16] en 2021 y a lo que llamaron modelos de difusión en cascada.

Un modelo de difusión en cascada se compone de una serie de modelos de

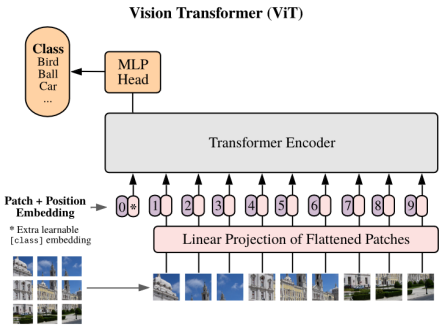


Figura 2.10: Vision Transformer (ViT). [7]

difusión condicionados tal y como se aprecia en la Figura 2.11. El primer modelo generará una muestra de baja resolución únicamente a partir de la condición y el resto generarán una muestra de mayor resolución a partir de la condición original y de la muestra obtenida por el modelo anterior. De esta forma se van añadiendo detalles de mayor resolución, lo que da como resultado una muestra final de alta resolución. Los modelos más conocidos de este tipo son Imagen [10] y DeepFloyd IF [4].

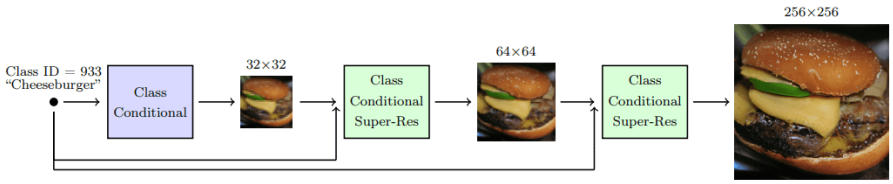


Figura 2.11: Pipeline de un modelo de difusión en cascada. [16]

Esta solución, aunque efectiva, es más costosa y requiere entrenar varios modelos. Como alternativa, en 2021, Rombach et al. [34] propusieron utilizar un VAE pre-entrenado para modelar un espacio de menor dimensión (espacio latente) y llevar a cabo los procesos de difusión y muestreo en ese espacio. Estos modelos se conocen como modelos de difusión latentes (LDMs, por sus siglas en inglés).

Estos LDM proyectan la entrada de alta dimensión $\mathbf{x}_0$ a un espacio latente $\mathbf{z}_0$ y aplican ahí el proceso de difusión, para posteriormente predecir $\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t)$ en lugar de $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$. De este modo, se puede obtener una muestra válida del espacio latente ($\mathbf{z}_0$) a partir de una muestra de ruido en ese espacio ($\mathbf{z}_T$). Después, la muestra generada $\mathbf{z}_0$ es decodificada a una imagen de mayor resolución $\mathbf{x}_0$.

Para un LDM con un codificador ε y una representación latente $\mathbf{z}_t$, la pérdida se define como:

$$L_{LDM} = \mathbb{E}_{\varepsilon(x),t,\epsilon}[||\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{z}_t, t)||^2]$$

El LDM más conocido y utilizado actualmente es Stable Diffusion [44]. La Figura 2.12 muestra una imagen generada con este modelo.

Figura 2.12: Imagen de alta resolución generada por un LDM [34]

Stable Diffusion es el modelo text-to-image más conocido y utilizado actualmente por varias razones. En primer lugar, es un modelo text-to-image preentrenado con miles de millones de imágenes capaz de generar imágenes de gran calidad. Además, fue el primer gran modelo de este tipo publicado en código abierto y con pesos públicamente disponibles. Esto es relevante porque entrenar este tipo de modelos está fuera del alcance de la mayoría de estudiantes e investigadores. Por último, está integrado dentro de la librería Diffusers [8], una librería diseñada para propósitos educativos y académicos que agiliza la implementación de modelos de difusión.

Stable Diffusion utiliza una U-Net compuesta de 4 bloques down, 1 bloque intermedio y 4 bloques up. Cada bloque down/up está constituido por 3 sub-bloques y 1 bloque de downscaling/upscaling. Cada sub-bloque está constituido por 1 bloque residual seguido de un ViT compuesto a su vez de un mecanismo *cross-attention*, un mecanismo *self-attention* y un bloque feed-forward. Cabe destacar que los ViT que se usan dentro de Stable Diffusion no dividen las imágenes en parches sino que usan cada píxel de la imagen convolucional como

token.

Stable Diffusion es un LDM por lo que los procesos de difusión y muestreo se realizan en una proyección de menor dimensionalidad de los datos. El modelo encargado de aplicar y revertir esta proyección será un VAE pre-entrenado. Si las imágenes tienen 64 píxeles de lado y 3 canales de color (64x64x3), el VAE las reducirá a 8x8x4.

Para soportar condicionamiento por texto, Stable Diffusion usa tanto el tokenizador como el codificador de texto de CLIP (τ_θ) para obtener los embeddings de cada token del texto de entrada. Esta representación vectorial del texto se combina con las imágenes convolucionales usando los mecanismos *cross-attention* de la U-Net. El paso t del proceso de difusión se codifica usando *positional encoding*. La Figura 2.13 muestra los componentes de Stable Diffusion y cómo se relacionan.

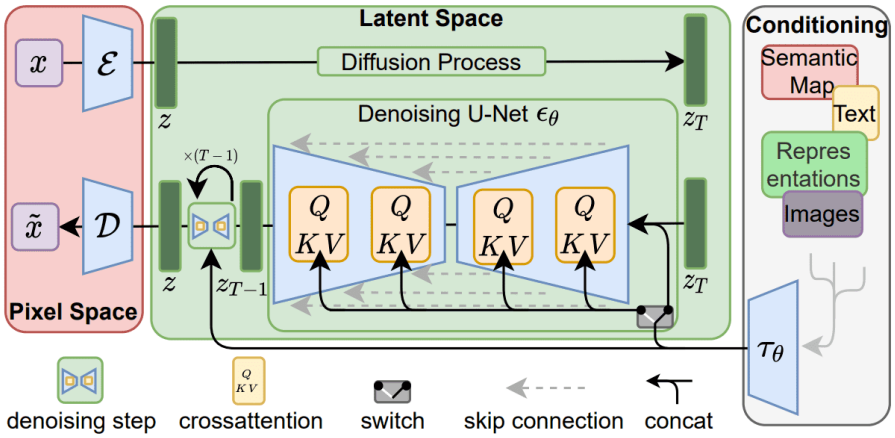


Figura 2.13: Latent diffusion model. [ref.]

2.5. LoRA

Los Transformers han sido la base para el desarrollo de Grandes Modelos del Lenguaje (LLMs, por sus siglas en inglés), modelos con un número masivo de parámetros entrenados sobre vastas cantidades de texto y capaces de generar respuestas coherentes y contextualmente relevantes a diversas consultas. Estos modelos han emergido como una de las herramientas más prometedoras y poderosas en el campo de la Inteligencia Artificial. Ejemplos notables de este tipo de modelos incluyen Bloom o GPT-4.

Sin embargo, cuando se trata de dominios específicos, aunque estos modelos pueden aprender nuevas tareas con unos pocos ejemplos (few-show learning), el re-entrenamiento del modelo sigue dando mejores resultados. Este re-entrenamiento resulta prohibitivo dado que el número de parámetros de estos modelos está en el orden de las decenas o cientos de miles de millones (175B parámetros en el caso de GPT-3).

En 2021, Hu et al. [18] propusieron Low-Rank Adaptation (LoRA), un método para hacer fine-tuning a LLMs pre-entrenados que permite conseguir resultados similares o incluso mejores a los que se obtendrían de entrenar el modelo completo pero reduciendo en varios órdenes de magnitud el número de parámetros entrenables. Según los autores, usando LoRA es posible re-entrenar GPT-3 reduciendo el número de parámetros entrenables en un factor 10.000x y la memoria gráfica requerida en un factor 3x.

La Figura 2.14 muestra qué capas componen estos módulos LoRA. Esencialmente, se suma a la salida de la capa lineal original la salida de 2 nuevas capas (A y B) siendo la dimensión que comparten muy reducida ($r \ll d_{model}$). A esta dimensión se le conoce cómo rango r y será un hiperparámetro específico de LoRA. Opcionalmente, la salida del módulo LoRA se puede escalar si queremos modular el grado en el que nuestro modelo re-entrenado se comporta como el modelo original. Esta aproximación modular, manteniendo los pesos originales, permite guardar por separado y reemplazar rápidamente los pesos de distintos módulos LoRA entrenados para tareas específicas.

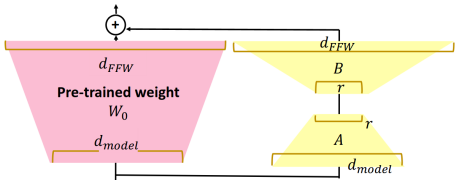


Figura 2.14: Low-Rank Adaptation (LoRA). [3]

En principio, se puede aplicar LoRA a cualquier capa de una red neuronal para reducir el número de parámetros entrenables. Los autores aplican LoRA en las capas de proyección de los mecanismos de atención de un Transformer aunque también se puede aplicar en mecanismos de atención usados en CNNs. La Figura 2.15 muestra cómo LoRA se puede aplicar en un mecanismo de atención.

LoRA se ha usado en modelos de difusión como Prolific Dreamer [45], un modelo presentado en mayo de 2023 capaz de generar modelos NeRF de alta calidad a partir de texto.

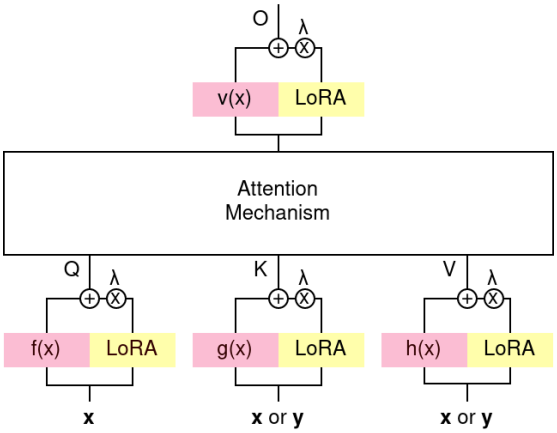


Figura 2.15: Aplicación de LoRA en un mecanismo de atención.

2.6. Condicionamiento

Un aspecto clave de la generación de imágenes es el control del proceso de muestreo para condicionar las muestras generadas con una etiqueta de clase o un vector de características extraído de una imagen o texto. Al proceso de muestreo guiado se le conoce como difusión guiada.

En 2021, Dhariwal & Nichol [6] propusieron un método de guiado que permitía guiar el proceso de muestreo de un modelo de difusión incondicional pre-entrenado. Este método, conocido como guiado por clasificador, utiliza un clasificador externo pre-entrenado $f_\phi(y|\mathbf{x}_t)$ para clasificar la muestra ruidosa $\mathbf{x}_t$. Después usan el gradiente del clasificador $\nabla_{\mathbf{x}_t} \log f_\phi(y|\mathbf{x}_t)$ para guiar el proceso de muestreo. De este modo la muestra va obteniendo una clasificación más cercana a y conforme avanza el proceso de muestreo.

Siendo s un parámetro para controlar la intensidad del guiado:

$$\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t) = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) - \sqrt{1-\bar{\alpha}_t} w \nabla_{\mathbf{x}_t} \log f_\phi(y|\mathbf{x}_t)$$

En 2022, Ho & Salimans [17] demostraron que es posible lograr difusión guiada sin usar un clasificador externo. Este guiado sin clasificador se logra entrenando un modelo de difusión condicional $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t|y)$ y otro incondicional $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t|\emptyset)$ utilizando la misma red neuronal. Durante el entrenamiento, los valores de la condición y de ciertas muestras elegidas aleatoriamente se reemplazan por ceros ($y = \emptyset$), haciendo que y sea irrelevante en el proceso de aprendizaje. De esta manera se expone el modelo a configuraciones condicionales e incondicionales.

Después, en el proceso de muestreo, el ruido estimado por la red condicionado por y para el paso t se obtiene como una interpolación lineal de la estimación condicionada por y y la estimación condicionada por $\emptyset$. La interpolación está escalada por el parámetro s que nos permite controlar la intensidad del condicionamiento:

$$\hat{\boldsymbol{\epsilon}}_\theta(\mathbf{x}_t, t, y) = s \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t|y) + (1 - s) \cdot \boldsymbol{\epsilon}_\theta(\mathbf{x}_t|\emptyset)$$

Este enfoque requiere estimar $\boldsymbol{\epsilon}_\theta$ dos veces por cada paso en el proceso de muestreo pero simplifica el entrenamiento al utilizar un solo modelo para el guiado.

Nichol et al. en 2022 [29], en su propuesta del modelo de difusión guiado GLIDE, exploraron ambas estrategias de guiado y concluyeron que el guiado sin clasificador ofrecía mejores resultados por lo que es el que utilizaremos en este trabajo.

Todos los modelos de difusión requieren que la red encargada de estimar $\boldsymbol{\epsilon}_\theta$ tenga en cuenta al menos el paso t del proceso de difusión. En el caso de usar guiado sin clasificador es necesario que la red también tenga en cuenta la condición y. Existen distintas formas de conseguir inyectar esta información para condicionar la salida de la red.

Como hemos visto, nuestro modelo estimará $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ lo que implica que la red no sólo tomará $\mathbf{x}_t$ como entrada sino que deberá ser condicionada, al menos, por t. Aprovechando que t indica un orden temporal, Ho et al. [15] utilizaron un embedding posicional sinusoidal como el que se usa en una red Transformer para codificar la posición de un token en una frase de tal forma que $\text{SinPosEmb}(t) = \vec{p}_t$. Después, en cada bloque residual de la U-Net, se proyecta $\vec{p}_t$ a un vector de tantas dimensiones como canales haya a la salida del bloque residual y se suma este vector a los canales de cada píxel de la salida de la primera capa convolucional del bloque residual. Este proceso se muestra en la Figura 2.16.

Si condicionamos por t podremos llevar a cabo el proceso de muestreo correctamente pero no podremos indicar al modelo que las muestras obtenidas cumplan alguna condición o restricción extra y. De acuerdo con Dhariwal et al. [6] es posible modelar distribuciones condicionales estimando $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, y)$.

Para condicionar por una clase y, los autores recomendaban usar *adaptative group normalization* (AdaGN), que funciona de forma similar al condicionamiento por t pero inyectando también una proyección de y en los bloques residuales tal y como se muestra en la Figura 2.17.

Condicionar por clase puede ser suficiente para algunas tareas pero es un condicionamiento muy limitado. En 2021, Nichol et al. [29] consiguieron condi-

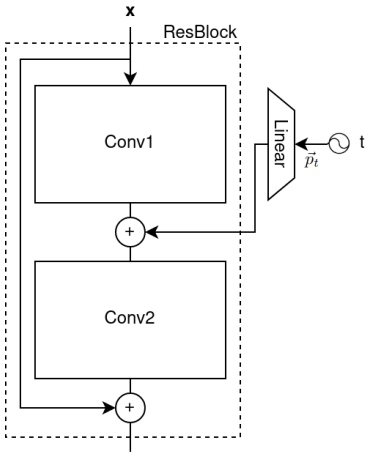


Figura 2.16: Condicionamiento por t.

cionar por texto un modelo de difusión. Este condicionamiento consistía en usar un Transformer, en concreto el codificador de texto de CLIP, para convertir el texto tokenizado a embeddings para después inyectarlo en los bloques residuales con AdaGN y, en paralelo, concatenar una proyección de estos embeddings a la salida de cada mecanismo *self-attention* de la red U-Net.

Los mecanismos *cross-attention* han sido efectivos en el aprendizaje de modelos basados en atención que combinan entradas de distintos dominios, lo que se conoce como modelos multimodales. Rombach et al. [34] propusieron utilizar estos mecanismos para permitir un condicionamiento multimodal. Tal y como se muestra en la Figura 2.18, después de cada bloque residual incluyen un mecanismo de atención donde Q será una proyección de la serie de píxeles de la imagen convolucional y K,V serán proyecciones de $\tau_\theta(y)$, donde τ_θ es un codificador específico del dominio de y. Si y es un texto, τ_θ será un Transformer como por ejemplo el codificador de texto del modelo CLIP. Este es el mecanismo de condicionamiento usado en Stable Diffusion.

En noviembre de 2021, Saharia et al. [36] presentaron Palette, un modelo de difusión capaz de resolver distintas tareas de traducción de imagen (restauración, colorización, inpainting y uncropping) sin cambiar la arquitectura ni los parámetros de entrenamiento. En este caso, siendo c una imagen con las mismas dimensiones que $\mathbf{x}_0$, consiguen que la red U-Net estime $\boldsymbol{\epsilon}(\mathbf{x}_t, t, c)$ concatenando c y $\mathbf{x}_t$ a la entrada de la red. Consiguen que el modelo lleve a cabo distintas tareas aplicando distintas máscaras que determinan, durante el proceso de muestreo, qué parte de $\mathbf{x}_{t-1}$ será sustituida por $\mathbf{c}_{t-1}$.

Un año después, Xu et al. [48] presentaron VersatileDiffusion, un modelo de difusión multimodal multitarea, capaz de realizar generación text-to-image,

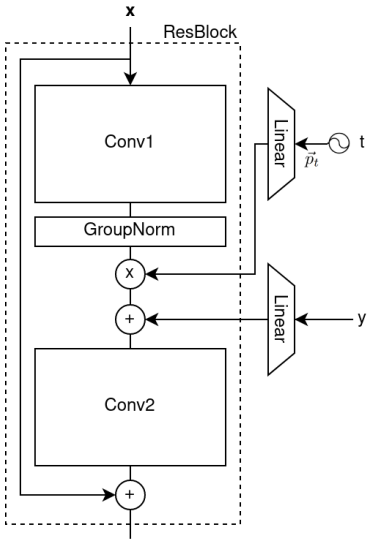


Figura 2.17: Condicionamiento por clase.

image-to-text, image-variation, text-variation, latent image-to-text-to-image editing, entre otras, todo ello sin requerir reentrenamiento. Consiguen esta versatilidad gracias a ciertas capas que pueden ser silenciadas en función del dominio de la entrada, de la condición o de la salida.

En febrero de 2023, Zhang et al. [53] propusieron ControlNet, una arquitectura que permite que modelos de difusión pre-entrenados condicionados por texto como Stable Diffusion soporten nuevos tipos de condicionamiento como bocetos, mapas de profundidad, keypoints o poses humanas. ControlNet clona los pesos de un gran modelo de difusión en una 'copia entrenable' y una 'copia bloqueada': la copia bloqueada preserva la capacidad de la red original y la copia entrenable se entrena con datos específicos de la tarea. Los bloques de la red entrenable y bloqueada están conectados con un tipo único de capa de convolución que los autores llaman *zero convolution* que consiste en una capa convolucional 1x1 donde el bias y los pesos están inicializados con ceros. Según los autores, entrenar estos modelos es tan rápido como hacer fine-tuning al modelo original. La Figura 2.19 muestra cómo se aplicaría ControlNet a un bloque o serie de capas de una red neuronal.

2.7. Datasets

Uno de los factores críticos para el entrenamiento y la evaluación efectiva de los modelos de difusión es la disponibilidad de datasets de imágenes de alta

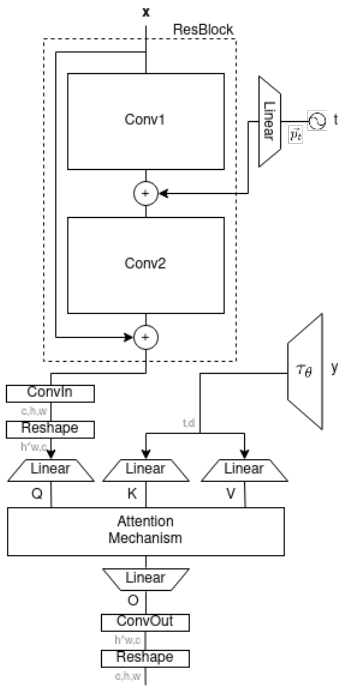


Figura 2.18: Condicionamiento multimodal.

calidad. En particular, cuando nos enfocamos en la generación de rostros humanos, varios datasets han emergido como los más utilizados en la comunidad investigadora debido a su gran tamaño y diversidad.

En 2007, Huang et al. presentaron Labeled Faces in the Wild (LFW) [19], un dataset compuesto por más de 13.000 imágenes de rostros, cada una etiquetada con 73 atributos.

En 2015, Liu et al. presentaron CelebFaces Attributes Dataset (CelebA) [26], un gran dataset con más de 200.000 imágenes de famosos, cada una con 40 anotaciones de atributos. Las imágenes de este dataset cubren grandes variaciones de pose y fondo. Este dataset es uno de los más utilizados en el ámbito de la generación de imágenes de rostros. Dada su diversidad y gran volumen, CelebA es frecuentemente utilizado para evaluar la capacidad de los modelos de difusión. Los autores de DDPM usaron, CelebA-HQ, una versión de más calidad de este dataset, para evaluar la calidad de su modelo de difusión.

En 2018, Karras et al. presentaron Flickr-Faces-HQ Dataset (FFHQ) [22], un dataset de 70.000 imágenes de caras de muy alta calidad ideado originalmente para entrenar redes GAN. Este dataset no incluye atributos o metadatos que describan cada imagen por lo que no podemos usarlo directamente para

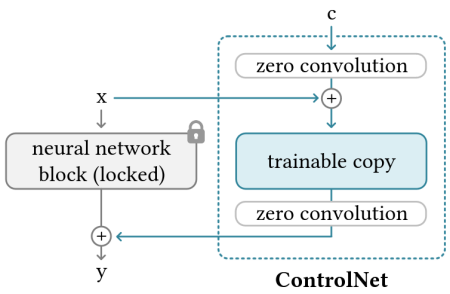


Figura 2.19: ControlNet. [53]

entrenar un modelo generativo condicional.

Si queremos generar imágenes a partir de texto necesitaremos datasets compuestos de parejas imagen-texto como Microsoft Common Objects in Context (MS COCO) [25], Conceptual Captions [39] o LAION-400M [38]. Los modelos text-to-image que constituyen actualmente el Estado del Arte han sido entrenados con este tipo de datasets. Sin embargo, todos estos datasets son genéricos: incluyen imágenes de distintos objetos, en distintos entornos e incluso con distintos estilos artísticos.

En 2021, Jiang et al. presentaron CelebA-Dialog [20], un dataset de parejas imagen-texto específico para caras y basado en el dataset CelebA. Este dataset fue ideado para entrenar una GAN en la tarea de edición de imágenes de caras por texto. Cabe destacar que las descripciones de las imágenes no capturan todos los atributos de CelebA, en su lugar las descripciones de cada imagen hacen referencia a 5 de los 40 atributos originales y tienen en cuenta la intensidad en la que se da cada atributo.

Capítulo 3

Generación condicional de caras

3.1. Modelos

Cumplir los objetivos de este trabajo implica implementar, entrenar y evaluar varios modelos de difusión condicionales. Dado que este trabajo tiene un objetivo académico y formativo, utilizaremos modelos de difusión más comunes y mejor documentados dentro de la tarea de síntesis de imagen: DDPM y Stable Diffusion.

3.1.1. DDPM

DDPM se ha probado efectivo en la tarea de generación incondicional de caras ya desde su presentación en 2020 [15]. Actualmente, los modelos que constituyen el Estado del Arte para esta tarea, en términos de FID obtenido para el dataset CelebA-64, son variaciones de este modelo. Sin embargo, estos modelos son incondicionales. En nuestro caso, modificaremos una implementación del DDPM original para que soporte condicionamiento por clase y guiado sin clasificador, algo que ya hicieron Dhariwal et al., obteniendo buenos resultados para CIFAR-10 e ImageNet [6].

El modelo estará constituido únicamente por una U-Net de 4 bloques down, 1 bloque intermedio y 4 bloques up. Cada bloque down/up está constituido por 2 bloques residuales y 1 bloque de downscaling/upscaling. Después de cada bloque residual se aplica un mecanismo *self-attention*. El condicionamiento por clase se aplica en cada bloque residual usando AdaGN.

Para su implementación hemos partido de un repositorio disponible en GitHub [2] en el que se implementa un DDPM incondicional usando PyTorch. Este código incluye dos clases principales: GaussianDiffusion, que implementa el plani-

ficador de ruido DDPM, y la clase U-Net que implementa la red encargada de estimar $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$.

Dentro de la clase U-Net, hemos modificado la implementación de los bloques residuales para que soporten un condicionamiento por AdaGN, recibiendo como entrada no sólo la imagen convolucional y una proyección del embedding de t sino también una proyección del vector de clases c. Hemos añadido parámetros necesarios como la dimensión esperada de c y la dimensión a la que queremos que sea proyectado.

Para implementar el guiado sin clasificador es necesario que, durante el entrenamiento, la condición sea irrelevante con cierta probabilidad. Además, para cada paso del proceso de muestreo, se deben estimar dos muestras de ruido, una aplicando la condición ($\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)$) y otra sin condición ($\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, \emptyset)$), para después hacer una interpolación lineal entre ambos resultados.

La red U-Net siempre espera recibir c, en el caso de hacer inferencia sin condición pondremos todos los valores de c a cero. Durante el entrenamiento hacemos esto antes de hacer inferencia a la red con un 10 % de probabilidad.

Después, en en cada paso del proceso de muestreo, duplicamos $\mathbf{x}_t$ y t, concatenamos c con un tensor de ceros de la misma dimensión, hacemos inferencia, separamos los resultados entre condicionados y no condicionados, y finalmente hacemos interpolación lineal para cada pareja condicionado-incondicionado. De esta manera sólo hacemos inferencia a la red una vez.

La librería Diffusers de HuggingFace [8] incluye para cada modelo de difusión una pipeline, un planificador de ruido y una red. La pipeline simplemente añade una capa de abstracción y permite a usuarios finales generar muestras sin saber la red o planificador de ruido que se esté usando.

Para poder tener un código unificado desde el que lanzar experimentos y evaluar resultados necesitamos esa capa de abstracción. Para ello hemos integrado este modelo en Diffusers adaptando la pipeline que esta librería ofrece para Stable Diffusion. Además, esto nos permite usar planificadores de ruido ya implementados en Diffusers. De hecho, para los experimentos realizados usamos DDPMScheduler (incluído en Diffusers) que sustituye a la clase GaussianDiffusion del repositorio original.

Finalmente, aunque AdaGN fue concebido para condicionamiento por clase, hemos comprobado que podemos condicionar por etiqueta usando esta misma técnica. Es decir, c no será un vector en formato 'one-hot' que indica una sola clase, sino que será una concatenación de vectores 'one-hot', donde cada uno indica la intensidad en la que se da cada atributo.

3.1.2. Stable Diffusion

En el caso de Stable Diffusion la implementación consistió en elegir el modelo pre-entrenado más adecuado, modificar el modelo para incluir los módulos LoRA y entrenar estos módulos para hacer fine-tuning al modelo original de forma eficiente.

Existen varios modelos pre-entrenados de Stable Diffusion. Los más utilizados son Stable Diffusion v1.4, v1.5 y v2.0 pero en todos ellos el VAE fue entrenado con imágenes de 512x512. Hemos comprobado que utilizar estos modelos no da buenos resultados para imágenes de 64x64 como las que usamos en este trabajo. Esto puede atribuirse a que, después de la codificación, en imágenes de 512x512 queda suficiente información para que el decodificador pueda restaurar la imagen original. Sin embargo, no queda suficiente información en el caso de imágenes de 64x64.

En su lugar, utilizaremos Stable Diffusion Nano v2.1 [11] el cual es equivalente a Stable Diffusion v2.0 pero utilizando un VAE entrenado con imágenes de 128x128. La Figura 3.1 muestra una comparativa de resultados obtenidos con Stable Diffusion y Stable Diffusion Nano.

(a) Stable Diffusion v2.0

(b) Stable Diffusion Nano v2.1

Figura 3.1: Efecto del VAE en Stable Diffusion.

Para su implementación usamos la librería Diffusers de HuggingFace. Stable Diffusion está oficialmente integrado en Diffusers y los pesos de sus distintas versiones se publican en HuggingFace.

Además, hacer fine-tuning a Stable Diffusion usando todos sus parámetros resulta prohibitivo por lo que es imprescindible usar LoRA. Al entrenar con LoRA reducimos el número parámetros entrenables de 865.911K a 830K, más de 1.000 veces menos.

Para implementar LoRA congelamos todos los parámetros, exploramos recursivamente los módulos de Stable Diffusion y a cada capa de proyección de cada mecanismo de atención (tanto *self-attention* como *cross-attention*) añadimos un módulo LoRA. De este modo, sólo se entrenarán los parámetros de los

módulos LoRA recién añadidos. Dentro de la librería Diffusers estos módulos se implementan con la clase LoRAAttnProcessor.

Como nota adicional, hemos observado que el VAE presenta una limitación para generar imágenes pequeñas con estos modelos, por lo que hemos considerado eliminar el VAE y realizar el proceso de muestreo directamente en el espacio de datos. Esta decisión invalida los pesos previamente entrenados de cualquier variante de Stable Diffusion, obligándonos a utilizar otros modelos o a entrenar uno desde cero.

Hemos considerado utilizar la primera etapa de DeepFloyd IF. Sin embargo, la optimización de este modelo estaba fuera de nuestras posibilidades debido a la limitación de recursos. Al momento de realizar estas pruebas, no existía ninguna implementación oficial ni guía sobre cómo implementar LoRA para este modelo.

En su lugar, hemos entrenado varias redes U-Net de distintos tamaños en la tarea de estimar $\boldsymbol{\epsilon}_\theta$ condicionado por texto, trabajando siempre en el espacio de los datos. El resultado de estas pruebas no ha sido satisfactorio porque modelar el condicionamiento por texto resulta mucho más costoso y serían necesarios tiempos de entrenamiento considerablemente más largos.

3.2. Métricas

3.2.1. Calidad de imagen

Los modelos generativos de imágenes se pueden evaluar por la calidad de las imágenes que producen o, en el caso de modelos generativos condicionales, por la correspondencia entre las imágenes producidas y la condición que se pasó al modelo, a lo que nos referiremos como calidad del condicionamiento. En ambos casos, la evaluación es inherentemente subjetiva por lo que las métricas utilizadas en la literatura utilizan modelos entrenados para aproximar el criterio humano.

Las métricas más utilizadas para evaluar la calidad de las imágenes son FID (Fréchet Inception Distance) e Inception Score (IS), ambas basadas en la red Inception, una CNN entrenada en el conjunto de datos ImageNet [5] y propuesta en 2014 por Szegedy et al. [42].

Inception Score (IS), fue propuesto por Salimans et al. en 2016 [37] para medir la calidad y diversidad de un conjunto de muestras. IS mide la calidad de las imágenes sintetizadas en base a las probabilidades de clase inferidas por una red Inception. Esencialmente, mide lo bien que encajan las imágenes sintetizadas en la distribución de imágenes reales tal y como las entiende la red Inception.

El principal problema de la métrica IS es que asume que las imágenes ge-

neradas serán de alguna de las clases del clasificador Inception por lo que su ámbito de aplicación es reducido. De hecho, esta condición no se cumple en las muestras que usaremos en este trabajo por lo que no tendremos en cuenta esta métrica.

En 2017, Heusel et al. [14] propusieron la métrica FID para medir la calidad de los modelos GAN y actualmente es la métrica más utilizada para medir la calidad de estos modelos. FID mide la distancia entre las representaciones de características de las imágenes reales y las sintetizadas en una capa concreta de una red Inception pre-entrenada. Al comparar las representaciones de características de las imágenes reales y sintetizadas en una capa concreta de la red Inception, FID capta el grado en que las imágenes sintetizadas son similares a las imágenes reales en términos de sus características de alto nivel.

FID emplea un modelo Inception previamente entrenado con ImageNet. A pesar de su vinculación con ImageNet, y a diferencia de la métrica IS (Inception Score), FID es adecuada para evaluar la calidad de modelos entrenados con otros conjuntos de imágenes. Este hecho destaca como la principal razón de su prevalencia en la literatura académica para evaluar la calidad de imágenes generadas por modelos generativos, incluyendo aquellos especializados en la generación de caras. En consecuencia, hemos seleccionado FID como métrica de calidad para este trabajo.

MTCNN es un detector de caras propuesto por Zhang et al. en 2016 [52]. Este modelo permite encontrar las regiones de una imagen en las que aparece una cara. Entre otros atributos, para cada cara ofrece un valor de confianza en su predicción. MTCNN podría usarse para evaluar la calidad de un modelo generativo específico de caras. Si un modelo generativo produce imágenes de caras tan realistas que MTCNN las puede detectar con alta confianza, esto podría considerarse un indicador de la calidad del modelo generativo.

En este trabajo, para evaluar la calidad de imagen usaremos las métricas FID (Fréchet Inception Distance) y MTCNN (Multi-task Cascaded Convolutional Networks).

Para implementar estas métricas, hemos recurrido a la extensión de PyTorch 'torchmetrics' [24]. Esta extensión ofrece la implementación de varias métricas, incluyendo FID y el modelo Inception pre-entrenado que se utiliza en la inferencia.

En la práctica, se acostumbra agrupar muestras reales y generadas para luego evaluar el FID entre pares aleatorios de cada conjunto. Sin embargo, con el fin de evitar medir el FID entre pares real-generado con atributos o condicionantes diferentes, optamos por emparejar imágenes reales y generadas que comparten la misma condición. Esta metodología ofrece resultados más confiables para un modelo condicional.

Para la implementación de MTCNN, hemos recurrido al paquete externo 'mtcnn-opencv' [27]. Este paquete incorpora el modelo y los pesos necesarios para realizar inferencias con MTCNN a partir de una imagen, utilizando internamente OpenCV.

3.2.2. Calidad de condicionamiento

Como hemos comentado, es posible hacer que un modelo de difusión genere muestras que cumplan alguna condición. Una pregunta crucial en el campo es cómo evaluar si las imágenes generadas satisfacen efectivamente las condiciones dadas.

En el caso de los modelos condicionados por texto (text-to-image) la métrica más usada es CLIP Score (CLIP-S), propuesta por Hessel et al. en 2021 [13]. CLIP-S mide la correspondencia entre una imagen y un texto. Mayor CLIP-S implica mayor compatibilidad. Esta correspondencia se mide como la similitud coseno entre los embeddings generados por el modelo CLIP para la imagen y su descripción.

CLIP es un modelo multimodal presentado en 2021 por Radford et al. [33] entrenado con 400 millones de parejas imagen-texto. Realmente, CLIP se divide en dos modelos: un codificador de texto y un codificador de imagen. Estos modelos convierten, respectivamente, un texto o una imagen a un vector de características o embedding con la propiedad de que los embeddings de una imagen y del texto que la describe serán muy parecidos. Una característica muy interesante de este modelo es que puede usarse como clasificador multipropósito sin reentrenamiento (zero-shot learning).

CLIP-S ha sido reconocida por sus autores como la métrica más fiable para evaluar la calidad del condicionamiento por texto y se ha establecido, de facto, como el estándar en la literatura académica para este propósito, por lo que será la que usemos en este trabajo. CLIP-S también está incluida en 'torchmetrics' [24].

En el caso de los modelos generativos de imágenes condicionados por clase, se suelen usar CNNs entrenadas para estimar la clase a partir de la imagen generada. Si la clase o atributos estimados por el clasificador concuerdan con la condición solicitada, se considera que el modelo condiciona correctamente. Usar como métrica la precisión de un clasificador específico de la tarea es algo que ya se ha hecho en otros trabajos importantes como Palette [36].

Hemos desarrollado y entrenado una CNN para la tarea específica de estimar la condición a partir de una imagen, esto es, analizar una imagen y estimar la intensidad de cada uno de sus atributos. Con el fin de complementar la métrica CLIP-S, usamos también la precisión de la estimación ofrecida por esta red

como métrica de calidad de condicionamiento.

Empleamos una red ResNet50 pre-entrenada con ImageNet y modificada para estimar la clase de cada atributo, donde cada clase representa la intensidad del atributo. Las características extraídas por la red se pasan como entrada al clasificador de cada atributo. Cada clasificador se compone de un dropout del 20 % seguido de una capa con tantas neuronas como niveles de intensidad deseamos predecir.

Como es un problema de clasificación hemos usado entropía-cruzada como función de pérdida haciendo la media de la pérdida obtenida para cada clasificador. Hemos usado Adam como optimizador y ReduceLROnPlateau para bajar la tasa de aprendizaje si la precisión en validación no aumenta después de 5 epochs.

Para el entrenamiento congelamos todos los parámetros excepto los de las capas de clasificación durante 5 epochs usando una tasa de aprendizaje de 0.001. Después descongelamos todos los parámetros y entrenamos el modelo completo durante 15 epochs bajando la tasa de aprendizaje a 0.0001.

Después de este entrenamiento, alcanzamos una precisión del 83 % sobre los datos de validación.

3.3. Dataset

Para nuestro análisis, hemos empleado el dataset CelebA, una amplia base de datos que contiene 202,599 imágenes de personas famosas. Las imágenes presentan una gran variedad de características faciales, como diferentes tipos de peinados, tonos de piel y expresiones faciales. La diversidad de este conjunto de datos CelebA y el hecho de que venga acompañado de atributos para cada imagen lo convierte en un candidato ideal para nuestro trabajo.

Con el objetivo de facilitar el entrenamiento, hemos seleccionado la versión centrada y recortada del conjunto de datos. En esta versión del dataset todas las caras aparecen centradas en la imagen y todas tienen un tamaño de 218x178 píxeles.

Usaremos el conjunto de atributos de CelebA Dialog que describe cada imagen de CelebA con 5 atributos, cada uno de los cuales tiene 6 posibles valores de intensidad. Estos atributos son 'flequillo', 'gafas', 'barba', 'sonrisa' y 'edad'. La Figura 3.2 muestra algunas imágenes de este dataset junto con sus atributos.

Además, cada imagen también viene acompañada con un texto que la describe a partir de esos 5 atributos y su intensidad. Las descripciones textuales contienen la misma información que los atributos en formato numérico, con la salvedad de que la descripción textual indica también el género de la persona.

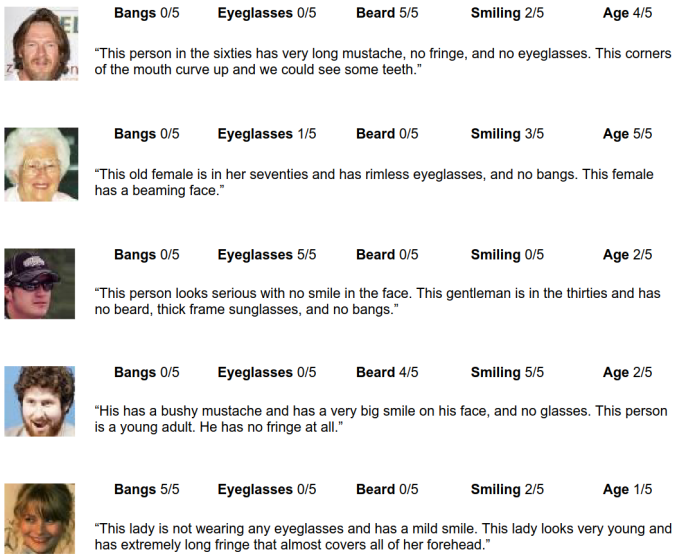


Figura 3.2: Muestras de CelebA Dialog

Hemos seguido la partición de datos proporcionada por los autores del dataset para mantener la integridad y comparabilidad de nuestros resultados. Según esta partición, el 80 % de las imágenes se destinaron al conjunto de entrenamiento, el 10 % al conjunto de validación, y el 10 % restante al conjunto de test.

3.4. Preprocesamiento

Teniendo en cuenta la exigencia de recursos computacionales de los modelos con los que vamos a trabajar, decidimos minimizar tal requerimiento generando imágenes de un tamaño reducido de 64x64 píxeles. Por ello, recortamos las imágenes originales para obtener una forma cuadrada y luego las escalamos a un tamaño de 64x64 píxeles.

Para fortalecer la robustez de nuestros modelos y aumentar sus capacidades de generalización, aplicamos técnicas de aumentado de datos. Este proceso consiste en crear nuevas muestras de entrenamiento a través de transformaciones aleatorias en el conjunto de datos original. Las transformaciones utilizadas se muestran en la Figura 3.3 e incluyen cambios en brillo, contraste, saturación, rotación, traslación y escalado. Han sido aplicadas tanto al entrenar los diferentes modelos de difusión como al entrenar el estimador de atributos.

Figura 3.3: Muestras de CelebA aplicando transformaciones aleatorias

Para implementar estas transformaciones, recurrimos a la librería Albumentations, que ofrece una amplia gama de técnicas de aumentado de datos.

3.5. Metodología

Dividiremos el experimento en dos partes principales. Primero, haremos una parametrización por separado para el modelo condicionado por atributos y para el modelo condicionado por texto hasta encontrar una configuración que nos permita maximizar la calidad de las muestras generadas para cada modelo. Segundo, haremos una comparativa entre el mejor modelo condicionado por atributo y el mejor modelo condicionado por texto.

Para ambos modelos haremos un entrenamiento de 5 epochs, suficiente para ver en qué caso se consigue una convergencia más rápida. Por limitaciones de hardware usaremos un tamaño de batch de 32. Comprobaremos el efecto de aplicar acumulación de gradiente (GA, por sus siglas en inglés) para simular tamaños de batch más grandes.

Aunque usar una tasa de aprendizaje más baja puede llevar a mejores resultados, hará que la convergencia sea más lenta lo que puede reducir, en las primeras epochs, las diferencias entre los resultados obtenidos para cada caso. Dado que en esta primera serie de experimentos sólo haremos pocas epochs estableceremos en todas las pruebas una tasa de aprendizaje inicial de 10^{-4}.

De acuerdo con la teoría, a mayor T más próxima será $q(\mathbf{x}_T)$ a una distribución Gaussiana isotrópica y mejor se ajustarán las muestras producidas a la distribución de los datos. Por esto, mediremos el efecto de este parámetro al entrenar el DDPM condicional.

Stable Diffusion Nano v2.1 fue entrenado para $T = 10^3$. Hacer fine-tuning

a este modelo con LoRA usando un valor distinto de T puede hacer que los pesos de los módulos LoRA entren en conflicto con los pesos pre-entrenados y congelados del modelo original. En cualquier caso, también variaremos este parámetro y comprobaremos si esta hipótesis es correcta.

Dado que queremos que los modelos soporten guiado sin clasificador, haremos que ambos modelos sean entrenados de forma incondicional un 10 % de las veces.

Para ambos modelos hemos implementado la estrategia de ponderación de loss Min-SNR-γ que según sus autores permite una convergencia más rápida. Vamos a evaluar también el efecto de usar esta estrategia.

Para el DDPM condicional, usaremos 256, 512, 768 y 768 canales para los 4 bloques down/up de la U-Net. Usar un número mayor de canales nos forzaría a reducir el tamaño de batch lo que ralentizaría demasiado cada entrenamiento. En su lugar, comprobaremos si podemos conseguir un modelo igual de efectivo pero usando menos parámetros.

En el caso de Stable Diffusion ajustado con LoRA, los pesos originales están congelados y no podemos cambiar el diseño de la red. En cambio, podemos modificar el hiperparámetro rango r de los módulos LoRA y comprobar si obtenemos los mejores resultados al usar $r = 4$, tal y como afirman los autores de LoRA.

En resumen, haremos un entrenamiento de cada modelo con cada posible combinación de los siguientes hiperparámetros:

- GA: No, 4
- T: $10^3, 10^4$
- Min-SNR-γ: No, Si (γ=5.0)
- (DDPM condicional) Núm. canales: [128, 256, 512, 512], [256, 512, 768, 768]
- (LoRA) r: 2, 4, 8

Después, entrenaremos cada modelo con su mejor combinación de parámetros durante 15 epochs.

Para la comparativa final generaremos 100 muestras por cada modelo, evaluaremos las muestras generadas usando las distintas métricas y compararemos la media de estas métricas para cada modelo.

Para poner en contexto los resultados, hemos evaluado cada métrica sobre un conjunto de muestras generadas por el modelo DDPM condicionado por

atributos antes del entrenamiento, inicializado con pesos aleatorios; y sobre un conjunto de muestras generadas por el modelo Stable Diffusion Nano 2.1 original, antes de añadir módulos LoRA. De este modo conocemos empíricamente la cota inferior de las métricas. De la misma manera, hemos evaluado cada métrica con muestras reales del conjunto de validación para conocer la cota superior. La Figura 3.4 muestra una imagen de cada grupo y la Tabla 3.1 expone los resultados obtenidos.

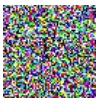
Metric	Untrained DDPM	SD Nano 2.1	Ground Truth
FID	387.9992	339.9721	0.0
MTCNN	0.83 %	20.25 %	99.91 %
CLIP-S	19.8594	22.9743	24.1421
Att. Estimator	70.60 %	70.40 %	82.85 %
			(ideal)

Cuadro 3.1: Evaluación de los modelo sin entrenar y de las muestras originales

Al evaluar sobre imágenes reales obtenemos FID igual a 0 porque las parejas de imágenes real-generada son idénticas por lo que las características obtenidas por el modelo Inception serán iguales y la distancia entre ellas será 0.

También cabe destacar que la precisión del estimador de atributos llega al 70 % incluso al recibir ruido porque el modelo estimará mínima intensidad para cada atributo, lo que resulta ser el caso más probable a excepción del atributo edad.

Por último, CLIP-S varía en un rango pequeño. Esto podría deberse a que las descripciones utilizadas no contienen información sobre el fondo, color de piel, ropa u otra información que llevaría a una mejor correspondencia con la imagen y, por tanto, a un mayor CLIP-S. Las descripciones utilizadas no buscan maximizar la correspondencia con la imagen sino que se limitan a describir 5 atributos y la intensidad en la que se presentan.



(a) Untrained DDPM

(b) SD Nano v2.1

(c) Ground Truth

Figura 3.4: Muestras de cada grupo

Capítulo 4

Experimentos y resultados

4.1. DDPM

La Figura 4.1 muestra algunas imágenes generadas con el modelo resultante de entrenar 5 epochs con la configuración que ha conseguido mejores resultados en términos de FID.

Figura 4.1: Muestras obtenidas con DDPM condicionado por atributos

La Tabla 4.1 muestra los mejores resultados obtenidos para cada métrica junto con los hiperparámetros del experimento en el que la métrica alcanzó ese valor.

El número de pasos del proceso de difusión T no afectó significativamente, llegando a obtenerse mejores resultados usando un menor T para varias de las métricas. Esto puede deberse a que el entrenamiento ha sido corto. Más pasos implica más distribuciones $p_\theta(\mathbf{x}_t, t)$ que deberán ser modeladas por la red. Es posible que se obtengan resultados de más calidad una vez estas distribuciones han sido modeladas pero converger hasta ese punto requeriría más epochs.

Metric	Best	Configuration			
		T	Min-SNR-γ	GA	Channels
FID	83.4184	1e+04	No	No	[256,512,768,768]
MTCNN	99.88 %	1e+03	5	No	[256,512,768,768]
CLIP-S	24.5876	1e+03	No	No	[256,512,768,768]
Att. estimator	88.20 %	1e+03	No	No	[256,512,768,768]

Cuadro 4.1: Mejores resultados obtenidos para cada métrica

Usar Min-SNR-γ no ha resultado en una diferencia notable. Los autores probaron esta estrategia de escalado de loss sobre el conjunto de datos CelebA 64x64 con un modelo incondicional y sobre el conjunto de datos ImageNet con un modelo condicional obteniendo mejoras en ambos. Sin embargo, indican que esta mejora es menor cuando la loss se calcula directamente comparando el ruido estimado $\boldsymbol{\epsilon}_\theta$, como es nuestro caso.

Acumular gradiente y simular un tamaño de batch mayor ha dado peores resultados. Esto puede deberse a que, cuando el tamaño de batch es mayor, el módulo del gradiente será menor y si mantenemos la tasa de aprendizaje el modelo convergerá más lentamente.

Finalmente y como era de esperar, usar una red más grande llevó a mejores resultados en todas las métricas. Cabe destacar que hemos identificado configuraciones que se acercan a los mejores resultados pero utilizando redes con menos parámetros. La Tabla 4.2 muestra una comparativa de los mejores resultados obtenidos para cada métrica en función del tamaño de la red utilizada.

Metric	Best for [128,256,512,512]	Best for [256,512,768,768]
FID	98.3362	83.4184
MTCNN	99.64 %	99.88 %
CLIP-S	24.3546	24.5876
Att. Estimator	85.80 %	88.20 %
		(+107K params.)

Cuadro 4.2: Mejores resultados en función del tamaño de la red

En cuanto a las medidas obtenidas, cabe destacar que alcanzamos unos valores de CLIP-S y precisión del estimador de atributos por encima del ground truth. Pensamos que esto puede explicarse con la presencia de etiquetas incorrectas o inconsistentes dentro del conjunto de muestras evaluadas.

El nivel de los atributos sigue por lo general un gradiente como puede ser la intensidad de la sonrisa o la longitud del flequillo. Es posible que dos personas presenten realmente el mismo nivel de un atributo pero, al convertir ese gradiente de intensidad en valores discretos, se haya asignado un nivel diferente a cada persona.

Si hay algunas imágenes que están mal etiquetadas, es probable que sus características no sean lo suficientemente comunes o destacadas como para que la red las adopte como representativas de ese atributo o de ese nivel específico. Por esto, la red llega a generar muestras correctamente condicionadas aunque las muestras de entrenamiento presenten algunos errores de etiquetado.

A la vista de los resultados obtenidos, elegimos entrenar durante más epochs un DDPM condicionado por atributos usando los siguientes hiperparámetros:

- GA: No
- T: 10^3
- Min-SNR-γ: No
- Núm. canales: [256, 512, 768, 768]

4.2. Stable Diffusion

La Figura 4.2 muestra algunas imágenes generadas con el modelo resultante de entrenar 5 epochs con la configuración que ha conseguido mejores resultados en términos de FID.

Figura 4.2: Muestras obtenidas con Stable Diffusion Nano v2.1 + LoRA

La Tabla 4.3 presenta los mejores resultados obtenidos para cada métrica, junto con los hiperparámetros del experimento en el que se logró cada uno de

esos valores.

Metric	Best	Configuration			
		T	Min-SNR-γ	GA	r
FID	144.8888	1e+04	No	No	2
MTCNN	99.97 %	1e+03	5	4	4
CLIP-S	25.3535	1e+03	5	No	8
Att. estimator	72.80 %	1e+03	5	No	4

Cuadro 4.3: Mejores resultados obtenidos para cada métrica

En este caso, el efecto de cada hiperparámetro ha sido menor en términos generales aunque podemos observar que algunos resultados son consistentes con los obtenidos para el DDPM condicionado por atributos: ni usar un mayor número de pasos T ni aplicar acumulación de gradiente ha llevado a mejores resultados. Sin embargo, usar Min-SNR-γ parece haber contribuido positivamente.

En cuanto al rango de los módulos LoRA, en la mejor configuración encontrada para 2 de las 4 métricas se usaba $r = 4$. Esto coincide con los resultados obtenidos por los autores de LoRA, quienes recomiendan usar $r = 4$.

Las imágenes producidas no muestran detalles de alta frecuencia lo que impacta de forma notable en la calidad y el realismo de las muestras generadas. Pensamos que esto se debe a que el VAE usado en este modelo (entrenado con imágenes de 128x128 píxeles), aunque mejora respecto al VAE de los modelos Stable Diffusion originales, sigue sin ser suficiente para reconstruir fielmente imágenes de 64x64 píxeles.

A la vista de los resultados obtenidos, entrenaremos durante más epochs los parámetros de los módulos LoRA añadidos a Stable Diffusion Nano 2.1 usando para ello los siguientes hiperparámetros:

- GA: No
- T: 10^3
- Min-SNR-γ: 5
- r: 4

4.3. Análisis de resultados

Las Figuras 4.3 y 4.4 muestran imágenes generadas con el DDPM condicionado por atributos y por Stable Diffusion Nano condicionado por texto, respectivamente. Cada modelo ha sido entrenado durante 15 epochs con la configuración

elegida en las pruebas anteriores.

Figura 4.3: Muestras obtenidas con DDPM condicionado por atributos

Figura 4.4: Muestras obtenidas con Stable Diffusion Nano v2.1 + LoRA

Podemos observar cómo el DDPM ha mejorado hasta ser capaz de generar imágenes más detalladas que Stable Diffusion (en adelante, SD). Sin embargo, si observamos la Tabla 4.4, veremos que SD+LoRA alcanza menor FID que el DDPM condicional, lo que parece indicar mayor calidad de imagen.

El modelo DDPM condicionado por atributos parece haber capturado las variaciones de saturación aplicadas a las muestras durante el entrenamiento como parte del aumentado de datos. Esto provoca que el color de las imágenes generadas esté con más frecuencia alejado del color correcto, lo que puede explicar el menor FID ya que, para calcular esta métrica, se usan parejas de imágenes real-generada donde la imagen real no tiene transformaciones ni variaciones de color aleatorias.

Además, el hecho de que entrenar durante más epochs los pesos de los módulos LoRA añadidos a SD Nano 2.1 no haya permitido al modelo generar carac-

terísticas de alta frecuencia parece confirmar que esta capacidad no es tanto de la red U-Net como del VAE, cuyos pesos no han sido modificados.

La métrica MTCNN ha llegado a una confianza muy alta en ambos casos. A partir de esta métrica podemos concluir que ambos modelos son capaces de generar caras y que los detalles y características de alta frecuencia de las imágenes no parecen influir significativamente en esta métrica.

También podemos ver como CLIP-S es ligeramente superior en SD+LoRA respecto a DDPM, lo que indicaría mejor calidad de condicionamiento. Sin embargo, el estimador de atributos consigue una precisión mayor para DDPM.

Métrica	Conditional DDPM	SD Nano 2.1 + LoRA
FID	104.5163	93.2055
MTCNN	99.97 %	99.89 %
CLIP-S	26.1205	26.3395
Att. Estimator	86.40 %	80.40 %

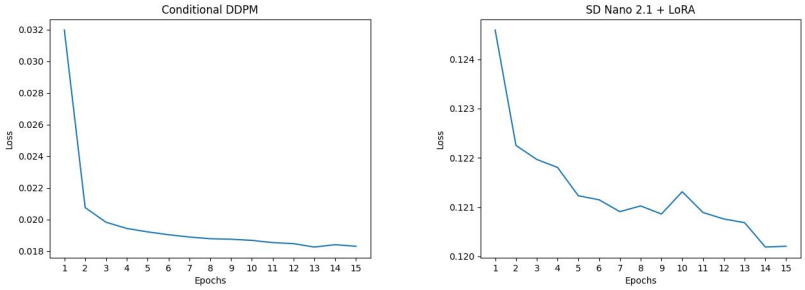
Cuadro 4.4: Resultados obtenidos al final del entrenamiento de cada modelo

La Figura 4.5 muestra imágenes generadas donde la condición incluye alguno de los 5 atributos en su máxima intensidad. Podemos ver cómo ambos modelos se ajustan correctamente a la condición. Además, por lo general, cuando se solicita un atributo con máxima intensidad ambos modelos parecen exagerar la condición llegando a ser más notable en las imágenes generadas que en las reales. Esto también puede contribuir al hecho de que se obtenga, para ambos modelos, mejores valores de CLIP-S y precisión del estimador de atributos que en la evaluación con muestras reales.

Figura 4.5: Imágenes con uno de los atributos en máxima intensidad.

En cuanto al entrenamiento podemos ver en la Figura 4.6 cómo la evolución

de la loss es más ruidosa en el caso de SD+LoRA. La función de pérdida que usamos se calcula comparando la salida de la red con $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ lo que provoca que la loss sea inherentemente ruidosa debido a su naturaleza estocástica. En el caso de SD+LoRA la loss ha variado menos durante el entrenamiento por lo que al ajustar el eje Y de la gráfica este ruido es más notable.



(a) Conditional DDPM (b) SD Nano 2.1 + LoRA

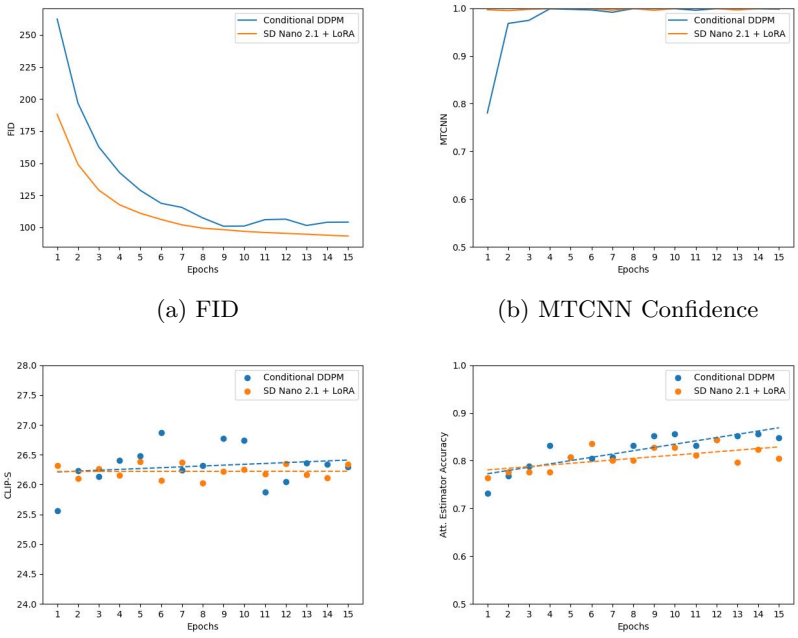
Figura 4.6: Evolución del entrenamiento

La Figura 4.7 muestra una comparativa de la evolución de cada métrica durante el entrenamiento. FID claramente indica una progresiva mejora en la calidad de imagen en ambos modelos, siendo esta mejora más acentuada en el caso de DDPM dado que el modelo no partía de pesos pre-entrenados. Ambos modelos logran generar caras con más del 90 % de confianza para MTCNN después de tan solo 2 epochs por lo que esta métrica no aporta mucha información pasado ese punto.

La precisión del estimador de atributos mejora ligeramente conforme progresa el entrenamiento. Concretamente, DDPM condicional pasa de un 73,2 % a un 84,8 % mientras que SD+LoRA pasa de un 76,4 % a un 80,4 %. CLIP-S por su parte muestra valores bajos desde el inicio del entrenamiento y no muestra ninguna mejora, lo que contradice la evaluación subjetiva que podemos hacer de la Figura 4.5. Por esta razón, la precisión del estimador de atributos resulta una métrica más fiable que CLIP-S para la tarea concreta que llevamos a cabo en este trabajo.

Como hemos visto, el guiado sin clasificador (CFG, por sus siglas en inglés) permite controlar el grado o la intensidad del condicionamiento manipulando el parámetro de escala s. Es habitual referirse a este hiperparámetro del proceso de muestreo como escala CFG o simplemente CFG.

Hemos evaluado el impacto que tiene variar la escala CFG tanto en la calidad de imagen como en la calidad de condicionamiento. Para ello hemos evaluado FID y la precisión del estimador de atributos para 250 muestras generadas por



(a) FID

(b) MTCNN Confidence

(c) CLIP-S

(d) Estimador de atributos

Figura 4.7: Evolución de las métricas durante el entrenamiento

cada modelo usando diferentes valores de escala CFG. La Figura 4.8 muestra las curvas de Pareto relacionando ambas métricas. Podemos ver como ambos modelos alcanzan una calidad de imagen similar en términos de FID pero el modelo DDPM condicionado por atributos se ajusta mejor al condicionamiento. Además, podemos ver que los valores de escala CFG entre 3 y 4 son los valores de escala CFG óptimos. Usar una escala CFG mayor mejora levemente el condicionamiento pero empeora significativamente la calidad de imagen.

La Figura 4.9 muestra una comparativa del número de parámetros de cada modelo, incluyendo también el modelo SD Nano 2.1 sin añadir los módulos LoRA. Dado que sólo entrenamos la U-Net, los pesos del codificador de texto y del VAE permanecen congelados durante todo el entrenamiento. Nótese que hacer fine-tuning a SD sin usar LoRA implicaría entrenar 865,91 millones de parámetros, más de 1000 veces más parámetros que usando LoRA. Por su parte, el modelo DDPM condicionado por atributos ha requerido entrenar 112,10 millones de parámetros, 7 veces menos que entrenando SD sin LoRA pero 135 veces más que usando LoRA.

Los entrenamientos se han llevado a cabo en un equipo con una GPU NVIDIA GeForce RTX 4080 (16Gb de memoria gráfica), 32 GB de RAM y un

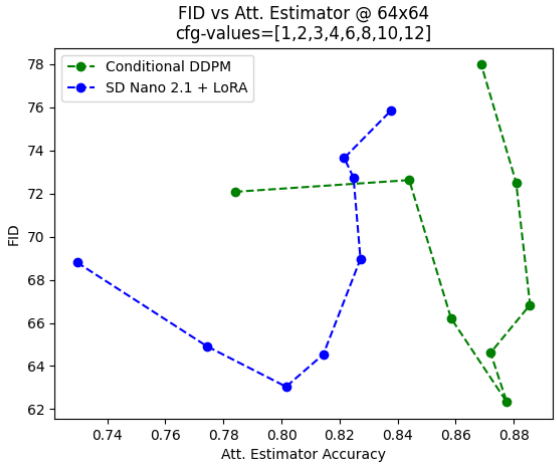


Figura 4.8: Curvas de Pareto.

procesador Intel(R) Core(TM) i7-13700KF de 16 núcleos. Cada epoch ha tomado 29 minutos en el caso del DDPM condicionado por atributos y 19 minutos en el caso de SD Nano 2.1 + LoRA.

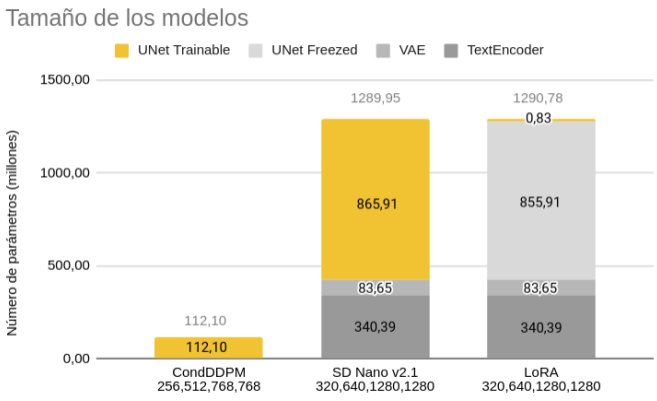


Figura 4.9: Comparativa del número de parámetros.

La segunda etapa del modelo DeepFloyd IF permite aumentar la resolución de imágenes de 64x64. A modo de prueba, hemos hecho inferencia a este modelo para algunas de las imágenes generadas. La Figura 4.10 muestra algunas de estas imágenes tras aplicar superresolución.

En resumen, para generar imágenes de caras de 64x64 píxeles a partir de una

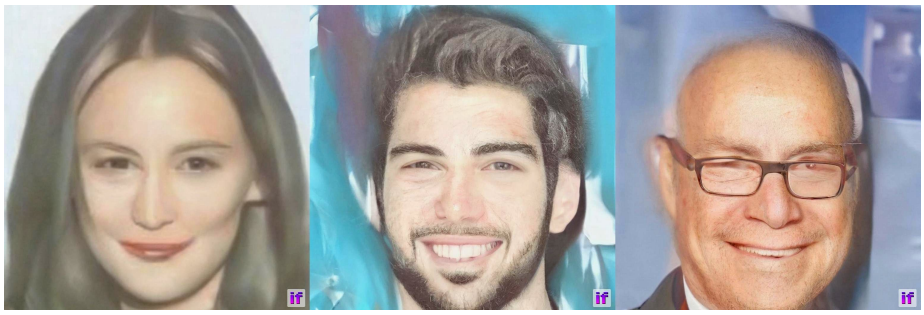


Figura 4.10: Imágenes generadas tras aplicar superresolución (256x256).

serie de atributos con distintos niveles de intensidad, usar un DDPM condicionado por atributos es una mejor aproximación que usar un DDPM condicionado por texto entrenado desde cero y mejor que ajustar con módulos LoRA un LDM pre-entrenado condicionado por texto cuyo VAE ha sido entrenado con imágenes de resolución 128x128 o superior.

Usar un DDPM condicionado por atributos es mejor que usar un DDPM condicionado por texto entrenado desde cero porque lograr la convergencia de este modelo requiere mayor número de muestras de entrenamiento, mayor número de parámetros y entrenamientos mucho más largos. Expresar el condicionamiento como texto implica utilizar $t \times d$ valores, donde t es el máximo número de tokens producido por el tokenizador y d la dimensión del vector de embeddings al que el codificador de texto convierte cada token. Sin embargo, usar condicionamiento por atributos implica utilizar $a \times i$ valores donde a es el número atributos e i son los distintos niveles discretos de intensidad de cada atributo. Aunque la información es la misma, expresar la condición como atributos requiere menos valores ($a \times i \ll t \times d$) lo que simplifica el problema y su modelización.

En el caso de ajustar un LDM pre-entrenado condicionado por texto, cuyo VAE ha sido entrenado con imágenes de resolución 128x128 o superior, el problema reside en el VAE en cuanto a que establece un límite en la mejora que se puede conseguir ajustando sólo los pesos de la red encargada de estimar $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t, c)$. Esta red trabaja en un espacio latente. Sin embargo, la capacidad de recuperar información perdida al proyectar los datos al espacio latente depende del tamaño de las muestras que se usaron al entrenar el VAE, la red encargada de proyectar y recuperar las muestras hacia o desde este espacio latente. Para generar imágenes de más calidad con este método habría que ajustar o re-entrenar el VAE con imágenes de 64x64 píxeles, o bien generar imágenes de al menos 128x128 píxeles.

Además, hemos comprobado que para esta tarea concreta resulta más fiable evaluar el condicionamiento usando como métrica la precisión de un modelo en-

trenado específicamente para estimar la condición usada a partir de la imagen. Una posible explicación sería que las descripciones textuales, aunque describen correctamente el nivel de los 5 atributos que podemos condicionar, no dan información que podría aumentar su CLIP-S como por ejemplo detalles sobre iluminación, color de piel, color del pelo o entorno en el que está la persona. Este problema podría resolverse haciendo fine-tuning al modelo CLIP con las muestras de entrenamiento y midiendo CLIP-S sobre el modelo CLIP ajustado. Sin embargo, esta solución haría que nuestras medidas de CLIP-S no fueran comparables con las de otros trabajos que usen el modelo CLIP original.

Capítulo 5

Conclusiones

En este trabajo, abordamos la generación condicional de imágenes de caras a una resolución de 64x64 píxeles. Hemos resuelto esta tarea usando dos modelos de difusión distintos: un modelo de difusión con condicionamiento por atributos y un modelo de difusión latente (LDM por sus siglas en inglés) condicionado por texto. Para el primer modelo adaptamos con éxito un DDPM incondicional para soportar condicionamiento por atributos así como guiado sin clasificador. Para el segundo modelo hacemos fine-tuning a una variante de Stable Diffusion usando LoRA, una novedosa técnica que permite hacer fine-tuning a este tipo de modelos de forma muy eficiente.

Stable Diffusion es un LDM condicionado por texto que representa el Estado del Arte en modelos text-to-image. Sin embargo, identificamos que el VAE de Stable Diffusion representa una limitación a la hora de generar imágenes pequeñas.

Desarrollamos un modelo capaz de estimar el nivel de intensidad de varios atributos de una cara a partir de una imagen. Hemos comprobado que la precisión de este estimador es una métrica de calidad de condicionamiento más fiable que CLIP-S para la tarea específica que abordamos en este trabajo.

En cuanto a los resultados obtenidos, encontramos que el aumento de T no produjo una mejora notable, al menos en las primeras epochs de entrenamiento. Tampoco hemos obtenido mejores resultados al usar acumulación de gradiente. La estrategia de escalado de loss Min-SNR-γ no mejoró el rendimiento del DDPM condicional, pero sí el de Stable Diffusion ajustado con LoRA. Finalmente, comprobamos que el uso de LoRA con $r = 4$ conduce a mejores resultados, lo que coincide con las recomendaciones de sus autores.

Para esta tarea específica obtenemos una similar calidad de imagen y un mejor condicionamiento utilizando un DDPM condicionado por atributos entrenado desde cero, en comparación con un LDM pre-entrenado condicionado

por texto, ajustado con LoRA y cuyo VAE ha sido entrenado con imágenes de resolución 128x128 o superior.

Como trabajo futuro, vemos diversas líneas de trabajo que podrían ayudarnos a mejorar y ampliar los resultados obtenidos:

- Dado que ambos modelos están integrados en la librería Diffusers y que se pueden usar diferentes planificadores de ruido sobre modelos de difusión pre-entrenados, se podría evaluar el efecto de usar otros planificadores de ruido en la calidad de imagen y de condicionamiento.
- Existen otros modelos de difusión condicionados por texto que realizan los procesos de difusión y muestreo en el espacio de los datos, como puede ser la primera etapa de DeepFloyd IF. Sin embargo, entrenar estos modelos es demasiado costoso. Sería interesante evaluar el desempeño de este tipo de modelos tras ser ajustados con LoRA para esta tarea específica.
- Utilizar aumentado de datos ha causado que el modelo genere imágenes con ligeras variaciones de color que pueden afectar negativamente al medir la calidad de la imagen con FID. Sería interesante evaluar hasta qué punto usar aumentado de datos es beneficioso para este tipo de modelos y para esta tarea concreta.
- Dado que el VAE de Stable Diffusion ha representado una limitación en la calidad de las imágenes generadas, sería interesante ajustar este VAE a las imágenes de nuestro dataset y comprobar si así se generan imágenes más detalladas.

Sin duda, el campo de los modelos de difusión sigue teniendo un vasto potencial para la investigación y la aplicación en diversos dominios. Esperamos que los resultados aportados en este trabajo se tengan en cuenta durante el diseño de futuros modelos de difusión.

Bibliografía

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[2] Tianqi Chen. Pytorch implementation of denoising diffusion probabilistic models. https://github.com/tqch/ddpm-torch, 2022.

[3] Cheng-Han Chiang, Yung-Sung Chuang, and Hung yi Lee. Recent advances in pre-trained language models: Why do they work and how to use them. https://d223302.github.io/AACL2022-Pretrain-Language-Model-Tutorial/, November 2022.

[4] DeepFloyd. Deepfloyd if. https://deepfloyd.ai/deepfloyd-if, 2023.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[6] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis, 2021.

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[8] Hugging Face. Diffusers: State-of-the-art diffusion pipelines. https://github.com/huggingface/diffusers, 2022.

[9] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[10] Google. Imagen. https://imagen.research.google, 2021.

[11] Bruno Guisard. Stable diffusion nano 2.1. https://huggingface.co/bguisard/stable-diffusion-nano-2-1, 2023.

[12] Tiankai Hang, Shuyang Gu, Chen Li, Jianmin Bao, Dong Chen, Han Hu, Xin Geng, and Baining Guo. Efficient diffusion training via min-snr weighting strategy, 2023.

[13] Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. Clipscore: A reference-free evaluation metric for image captioning, 2022.

[14] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[15] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[16] Jonathan Ho, Chitwan Saharia, William Chan, David J. Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation, 2021.

[17] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance, 2022.

[18] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models, 2021.

[19] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[20] Yuming Jiang, Ziqi Huang, Xingang Pan, Chen Change Loy, and Ziwei Liu. Talk-to-edit: Fine-grained facial editing via dialog. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2021.

[21] Yunfan Jiang. Elbo - what & why, Jan 2021.

[22] Tero Karras, Samuli Laine, and Timo Aila. Flickr-faces-hq dataset (ffhq). https://github.com/NVlabs/ffhq-dataset, 2018.

[23] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2022.

[24] Lightning-AI. Torchmetrics - machine learning metrics for pytorch. https://github.com/Lightning-AI/torchmetrics, 2022.

[25] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[26] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[27] mtcnn opencv. Mtcnn-opencv. https://github.com/egcode/mtcnn-opencv, 2021.

[28] Alex Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models, 2021.

[29] Alex Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob McGrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models, 2022.

[30] Mehrdad Noori, Ali Bahri, and Karim Mohammadi. Attention-guided version of 2d unet for automatic brain tumor segmentation. In *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*, pages 269–275, 2019.

[31] Ryan O'Connor. Introduction to diffusion models for machine learning, Apr 2023.

[32] William Peebles and Saining Xie. Scalable diffusion models with transformers, 2023.

[33] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021.

[34] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, June 2022.

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[36] Chitwan Saharia, William Chan, Huiwen Chang, Chris A. Lee, Jonathan Ho, Tim Salimans, David J. Fleet, and Mohammad Norouzi. Palette: Image-to-image diffusion models, 2022.

[37] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[38] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. Laion-400m: Open dataset of clip-filtered 400 million image-text pairs, 2021.

[39] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2556–2565, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[40] Vaibhav Singh. An in-depth guide to denoising diffusion probabilistic models – from theory to implementation, Mar 2023.

[41] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics, 2015.

[42] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[43] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[44] CompVis Computer Vision and Learning LMU Munich. Stable diffusion. https://huggingface.co/CompVis/stable-diffusion, 2022.

[45] Zhengyi Wang, Cheng Lu, Yikai Wang, Fan Bao, Chongxuan Li, Hang Su, and Jun Zhu. Prolificdreamer: High-fidelity and diverse text-to-3d generation with variational score distillation, 2023.

[46] Lilian Weng. From autoencoder to beta-vae, Aug 2018.

[47] Xiao Xiao, Shen Lian, Zhiming Luo, and Shaozi Li. Weighted res-unet for high-quality retina vessel segmentation. In *2018 9th International Conference on Information Technology in Medicine and Education (ITME)*, pages 327–331, 2018.

[48] Xingqian Xu, Zhangyang Wang, Eric Zhang, Kai Wang, and Humphrey Shi. Versatile diffusion: Text, images and variations all in one diffusion model, 2023.

[49] Chushu Yang, Xutao Guo, Tong Wang, Yanwu Yang, Nan Ji, Deling Li, Haiyan Lv, and Ting Ma. Automatic brain tumor segmentation method based on modified convolutional neural network. In *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 998–1001, 2019.

[50] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks, 2019.

[51] Jianxin Zhang, Zongkang Jiang, Jing Dong, Yaqing Hou, and Bin Liu. Attention gate resu-net for automatic mri brain tumor segmentation. *IEEE Access*, 8:58533–58545, 2020.

[52] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, oct 2016.

[53] Lvmin Zhang and Maneesh Agrawala. Adding conditional control to text-to-image diffusion models, 2023.