Implementación y análisis de Redes Neuronales Generativas mediante técnicas de Difusión

Álvaro Luna Ramírez
alvaroluna@correo.ugr.es

Pedro Jiménez García-Ligero
pedrojgl@correo.ugr.es

Germán José Padua Pleguezuelo
germanpadua@correo.ugr.es

Abstract

En este proyecto realizamos un estudio de los Modelos de Difusión aplicados a la generación de imágenes, enfocando nuestra investigación en la síntesis de automóviles a partir del "Stanford Cars Dataset". Mediante la adaptación y modificación de una arquitectura preexistente, conseguimos producir representaciones de vehículos en resolución 32x32. A esta implementación le realizamos una serie de modificaciones en las capas de la arquitectura, el algoritmo de entrenamiento, el valor de ciertos hiperparámetros y pruebas con distintas funciones de pérdida. Además, implementamos el algoritmo para calcular el FID score, con el que hemos evaluado y comparado nuestras distintas versiones del modelo. Los resultados empíricos destacan que el uso de la función de pérdida de Error Absoluto Medio supera al Error Cuadrático Medio en la generación de imágenes, como lo demuestra la mejora en los puntajes FID y la calidad visual de las imágenes generadas.

1.. Introducción

En las últimas décadas, los avances en el campo de la inteligencia artificial han revolucionado la capacidad de los ordenadores para comprender y generar datos complejos. Dentro de este panorama surgen los modelos generativos, que a través de un adecuado entrenamiento, son capaces de çrearïnformación nueva y realista. Existen distintos tipos de modelos generativos, como pueden ser las Redes Generativa adversativas (GAN), Variational autoencoder (VAE), Flow-based models y los Modelos de Difusión.

Los Modelos de Difusión son un tipo de modelo generativo que ha ganado una significativa popularidad en los últimos años. Desde el año 2020 han salido diferentes papers [1, 5] que demuestran el potencial de los Modelos de Difusión, capaces de superar a las Redes Generativas Adversarias en la síntesis de imágenes. Por ejemplo, cabe destacar el uso de los Modelos de Difusión en DALL-E 2, el modelo de generación de imágenes de OpenAI.

El objetivo de nuestro proyecto es crear imágenes de coches usando una red neuronal. Usaremos una U-Net basada en un Modelo Probabilístico de Difusión para entrenar el proceso de eliminación de un paso de ruido de imágenes de coches con una cierta cantidad de ruido añadido y entonces generar un coche realista a partir de una imagen que contenga únicamente ruido. La entrada al modelo será una imagen y un valor de paso, por lo que el proceso de predicción será el resultado de ir eliminando sucesivamente ruido a la imagen introducida, es decir, de ir eliminando paso a paso la cantidad de ruido restante.

2.. Background

2.1.. Modelos de difusión

Los Modelos de Difusión están inspirados en la termodinámica del no equilibrio. En esencia, operan al introducir de manera progresiva ruido Gaussiano en los datos de entrenamiento, para luego aprender a recuperar los datos revirtiendo este proceso de añadir ruido. Una vez completado el entrenamiento, podemos utilizar el Modelo de Difusión para generar nuevos datos simplemente pasando ruido muestreado aleatoriamente a través del proceso de *desruido* aprendido.

De manera más detallada, un Modelo de Difusión es un modelo de variable latente que se proyecta en el espacio latente a través de una cadena de Markov constante. Esta cadena añade ruido a los datos de forma progresiva para obtener la posterior aproximada $q(x_{1:T}|x_0)$, donde $x_1, \ldots, x_T$ son las variables latentes con la misma dimensionalidad que x_0.

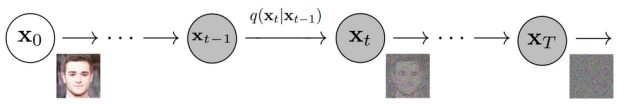


Figura 1. Cadena de Markov

La imagen se transformará gradualmente en ruido Gaussiano de manera asintótica. El objetivo de entrenar un Modelo de Difusión es aprender el proceso inverso, es decir, entrenar $p_\theta(x_{t-1}|x_t)$. Al seguir esta cadena en dirección contraria, tenemos la capacidad de generar datos novedosos:

pasaremos de imágenes de puro ruido a imágenes concretas (en nuestro caso, de automóviles).

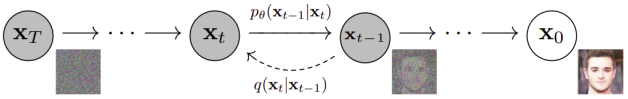


Figura 2. Proceso inverso

Los Modelos de Difusión no requieren 'Adversarial Training', que es una técnica utilizada en el aprendizaje profundo que involucra la creación de dos modelos neurales, conocidos como el 'generador' y el 'discriminador', que se entrenan simultáneamente en un proceso competitivo. Dicha técnica, que se utiliza por ejemplo en las GAN comentadas anteriormente, suele llevar a dificultades con la estabilidad del entrenamiento y la generación de datos no diversos. En cuanto a la eficiencia de entrenamiento, los Modelos de Difusión también presentan las ventajas añadidas de escalabilidad y paralelización.

2.2.. Probabilidad en los Modelos de Difusión

Como se ha indicado anteriormente, un Modelo de Difusión consta de:

1. Un proceso hacia adelante (proceso de difusión), en el cual a una imagen se le añade ruido progresivamente.

2. Un proceso hacia atrás (proceso de difusión inverso), en el cual el ruido se transforma de nuevo en una muestra de la distribución objetivo.

Para las siguientes definiciones, nos basamos en [4, 7].

2.2.1. Proceso de difusión

Dado un punto tomado de una distribución de datos $x_0 \sim q(x)$. Podemos definir un proceso hacia adelante en el que añadimos una pequeña cantidad de ruido Gaussiano a nuestro punto en T pasos, produciendo una secuencia de puntos ruidosos $x_1, \ldots, x_T$. Combinar este hecho con la suposición de Markov conduce a una parametrización simple del proceso hacia adelante. Para pasar del paso t al paso $t+1$ tenemos que:

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

Por lo que para realizar el proceso de difusión completo, nos quedaría:

$$q(x_{1:T}|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1})$$

Donde $\beta_1, \ldots, \beta_T$ es el ratio de difusión.

2.2.2. Proceso de difusión inverso

La esencia fundamental de los Modelos de Difusión reside en el proceso inverso. Si pudiéramos revertir el proceso de difusión y muestrear $q(x_{t-1}|x_t)$, podríamos recrear una verdadera muestra de una entrada de ruido Gaussiano, $x_T \sim \mathcal{N}(0, \mathbf{I})$. Sin embargo, no podemos estimar fácilmente $q(x_{t-1}|x_t)$, ya que esto necesitaría usar todo el dataset. Por tanto, necesitamos aprender un modelo p_θ para aproximar estas probabilidades condicionadas.

$$p_\theta(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \boldsymbol{\mu}_\theta(x_t, t), \boldsymbol{\Sigma}_\theta(x_t, t))$$

Luego el proceso de difusión inverso completo para obtener una muestra sería:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1}|x_t)$$

Aquí podemos ver un ejemplo de lo explicado. Esta sería la aplicación de un proceso de difusión a una imagen y el proceso inverso para recuperarla.

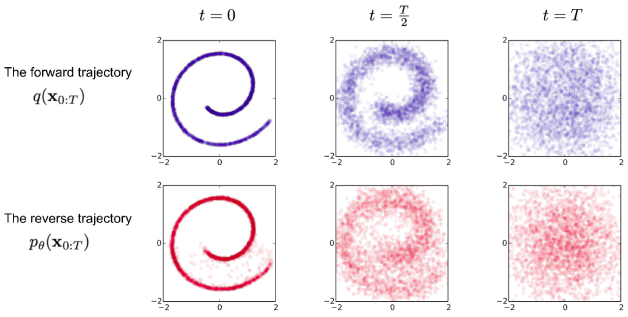


Figura 3. Ejemplo de entrenamiento de un Modelo de Difusión

3.. Related Works

Se han publicado distintos papers centrados en los Modelos de Difusión que demuestran su potencial como modelo generativo.

En [4] se introducen por primera vez los Modelos de Difusión Probabilística. Se basaron en la física estadística del no-equlibrio y la idea era destruir la estructura de la distribución de los datos lentamente a través de un proceso de difusión iterativo. Después, se aprendía a revertir dicho proceso, de forma que se recuperase la estructura de los datos. Esta nueva forma de definir modelos probabilísticos permitía una extrema flexibilidad en la estructura y un muestreo exacto.

En [5] se presentan resultados de generación de imágenes de alta calidad utilizando Modelos de Difusión. Se encontraron conexiones entre los Modelos de Difusión e inferencia variacional para entrenar cadenas de Markov.

En [1] introducen una serie de modificaciones que hicieron que los Modelos Probabilísticos de Difusión pudieran

alcanzar valores de *log-likelihood* competitivos, manteniendo una alta calidad de muestreo. Dichos cambios permitían también obtener resultados de estos modelos con muchos menos pasos.

4.. Métodos

4.1.. U-Net

Vamos a implementar una Red Neuronal Convolucional utilizando la arquitectura U-Net. Esta estructura, previamente discutida en las clases de la asignatura, es ampliamente utilizada en tareas de segmentación de imágenes y ha ganado considerable popularidad en los últimos años. Se compone de una arquitectura 'Codificador-Decodificador', que le confiere su característica forma de 'U'. Incorpora conexiones residuales que permiten fusionar las características del codificador (información espacial) con las del decodificador (información semántica de alto nivel). Esta arquitectura ha demostrado obtener resultados exitosos, incluso cuando se trabaja con conjuntos pequeños de imágenes.

En concreto, esta es la arquitectura U-Net que tendrá nuestro modelo:

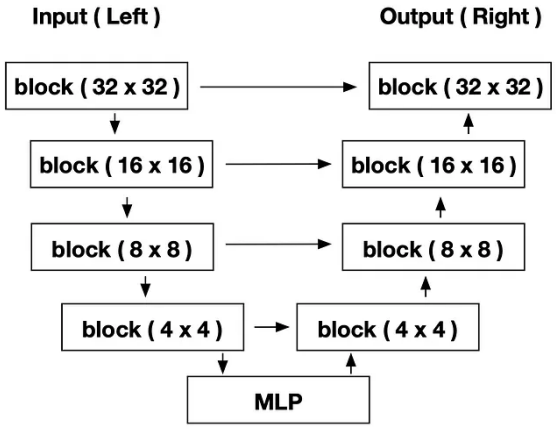


Figura 4. Arquitectura U-Net

4.2.. Modelo utilizado

En este apartado se detallará el modelo que hemos implementado. Cada bloque consta de 2 capas convolucionales con un parámetro de tiempo, lo que habilita a la red para determinar su paso de tiempo actual y extraer la información relevante. En la ilustración siguiente, se presenta el flujo del bloque, donde x_{img} representa la imagen de entrada (ruidosa) y x_{ts} corresponde a la entrada del paso de tiempo.

La estructura del modelo sigue el diseño de una U-Net, con un camino descendente para capturar contextos y un camino ascendente para localizar precisamente las características.

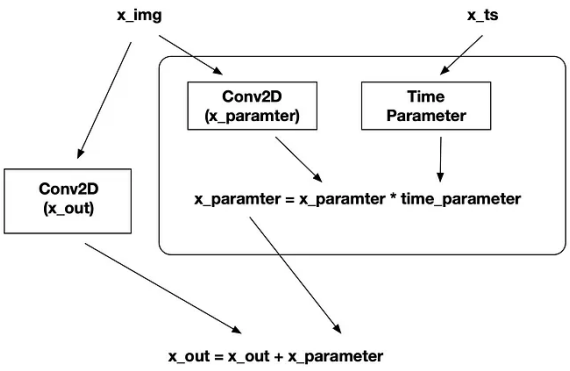


Figura 5

Inicialización:

- Secuencia de transformaciones para el parámetro de tiempo `l_ts`.
- 4 bloques que disminuyen progresivamente el tamaño: `down_x32`, `down_x16`, `down_x8` y `down_x4`.
- Perceptrón multicapa (MLP): Es una red neuronal compuesta por múltiples capas de neuronas interconectadas, en las que las salidas de las neuronas de una capa se convierten en entradas para la siguiente capa. Utiliza funciones de activación para introducir no linealidades y es capaz de aprender patrones complejos en datos, siendo comúnmente empleado en tareas de clasificación y regresión. Su entrenamiento puede ser computacionalmente costoso.
- 4 bloques para la parte de expansión: `up_x4`, `up_x8`, `up_x16` y `up_x32`.

Propagación:

- Transformamos el parámetro de tiempo para ser usado en el resto del modelo.
- Downsampling: Aplicamos los bloques para procesar la imagen de entrada. Después de cada bloque (excepto el último), se aplica un `max_pool2d` para reducir las dimensiones espaciales.
- MLP: Remodela la salida y se concatena con el parámetro de tiempo procesado, pasando este resultado a través de las capas densas. Esto permite que el modelo combine características espaciales con información temporal.

- Upsampling: Se realizan operaciones inversas al downsampling usando bloques para aumentar las dimensiones espaciales de la imagen. En cada paso, se concatena la salida correspondiente de la parte descendente con la salida actual, permitiendo que la red recupere información espacial perdida durante el downsampling. Utilizamos la función `F.interpolate` para escalar las imágenes a un tamaño mayor.

- Salida: Aplicamos una convolución para transformar la salida multidimensional en una salida de 3 canales (RGB).

Generación: El proceso de generación de imágenes de coche consiste en una concatenación de predicciones del modelo. A continuación, se muestra el algoritmo seguido:

- El modelo recibe como entrada una imagen de ruido aleatorio Gaussiano y el valor $t = 0$.

- Obtenemos una predicción del modelo, es decir, obtenemos la imagen resultante de revertir un paso el proceso de difusión de la imagen de entrada y el tiempo inicial.

- Introducimos al modelo la imagen obtenida y el paso de tiempo $t = 1$ para volver a obtener una predicción.

- Repetimos este proceso iterativo hasta llegar al último paso de tiempo definido, en nuestro caso 15. De esta manera, esta última predicción del modelo será el coche que se ha generado a partir de simplemente ruido.

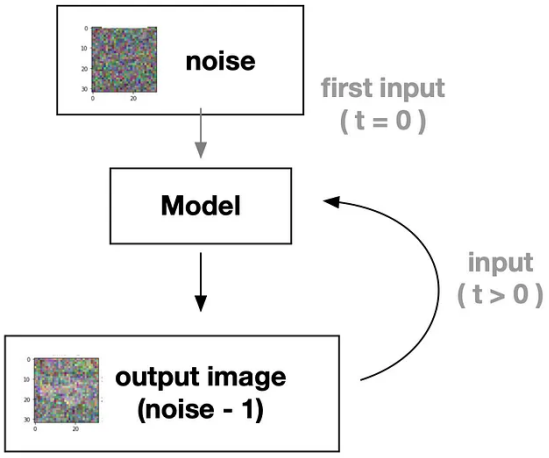


Figura 6. Proceso iterativo para generar la imagen de un coche a partir de ruido

Lo podemos ver también de esta forma para que resulte más claro:

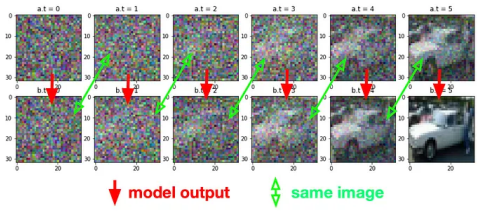


Figura 7. Flujo de la imagen en el modelo de difusión

5.. Experimentos

5.1.. Dataset

El dataset "Stanford Cars Dataset" consiste en 16.185 imágenes de 196 clases de coches. Estos datos fueron originalmente generados por el laboratorio de Inteligencia Artificial de la Universidad de Stanford para el desarrollo del paper ***3D Object Representations for Fine-Grained Categorization*** [6]. Las imágenes no están normalizadas a una resolución específica, oscila entre 200x200 píxeles y 600x600 píxeles. Es importante destacar que se obtuvo del sitio web Kaggle.

5.2.. Métricas de evaluación

Con el fin de evaluar de manera cuantitativa el desempeño del modelo, se han empleado tres de las métricas más convencionales en el ámbito de la generación de imágenes mediante Deep Learning: MAE MSE y FID.

5.2.1. Error Absoluto Medio (MAE):

$$\text{MAE}(x, y) = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i|$$

Esta métrica evalúa la diferencia promedio entre los valores predichos y los observados. A diferencia del Error Cuadrático Medio (MSE), el MAE es menos susceptible a ser influenciado por valores atípicos, debido a que que no eleva al cuadrado las diferencias. Esto lo hace particularmente valioso en contextos donde las predicciones pueden ser visualmente precisas pero varían en intensidad de píxeles, ya que el MAE proporciona una evaluación directa y robusta de la magnitud absoluta de los errores. En general, el MAE puede ser útil para evaluar aspectos específicos de las imágenes generadas, especialmente cuando se analiza la robustez del modelo frente a variaciones en la intensidad de píxeles.

5.2.2. Error Cuadrático Medio (MSE):

$$\text{MSE}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(x_i - y_i)^2$$

Esta métrica cuantifica la discrepancia cuadrática media entre los valores predichos y los reales, destacándose por su capacidad de penalizar errores grandes. El MSE es particularmente efectivo para identificar áreas donde el modelo puede tener dificultades en generar detalles específicos. Su cálculo sencillo y su interpretación intuitiva lo convierten en una métrica fundamental para evaluar la calidad global de las imágenes generadas.

5.2.3. Frechet Inception Distance (FID):

Sean $\mathbf{X} \sim \mathcal{N}(\mu, \Sigma)$ e $\mathbf{Y} \sim \mathcal{N}(\mu', \Sigma')$ dos distribuciones Gaussianas multidimensionales. [3] Definimos la distancia de Frechet de $\mathbf{X}$ e $\mathbf{Y}$ como:

$$d_F(\mathbf{X}, \mathbf{Y})^2 = \|\mu - \mu'\|_2^2 + \operatorname{tr}\left(\Sigma + \Sigma' - 2\left(\Sigma\Sigma'\right)^{\frac{1}{2}}\right)$$

El FID es esencial en la evaluación de modelos generativos para la generación de imágenes. Mide la distancia entre las distribuciones de características de las imágenes generadas y las reales, utilizando una red neuronal como Inception. [8] A diferencia de las métricas anteriores, el FID valora no solo la similitud píxel a píxel, sino también la calidad y diversidad de las características semánticas. Esto permite una evaluación más completa del realismo y variabilidad de las imágenes creadas. Además, exhibe una menor sensibilidad a artefactos locales, lo que contribuye a que sea un indicador robusto de la calidad visual global.

Importante destacar, en nuestro caso, las imágenes del dataset son de tamaño 32x32. Para calcular el FID de manera efectiva, estas imágenes deben ser reescaladas a un tamaño de 299x299, el tamaño estándar que requiere el modelo Inception v3.

Para calcular el FID, utilizamos una versión adaptada del modelo Inception v3, entrenado en ImageNet pero sin la capa de clasificación final. De esta forma, obtenemos el vector de activación de tamaño 2048 de su última capa pool. Este proceso se realiza tanto para cada batch de test del dataset como para un conjunto de predicciones del modelo. A partir de aquí, calculamos las medias y covarianzas de las estadísticas de activación correspondientes para construir las distribuciones normales $\mathcal{N}(\mu, \Sigma)$ y $\mathcal{N}(\mu', \Sigma')$, y finalmente, calcular la distancia FID.

Adicionalmente, dado que el cálculo de esta métrica no está disponible en las librerías estándar de Pytorch, hemos integrado a nuestro proyecto un código adaptado del siguiente repositorio en GitHub.

Hemos decidido incluir FID en lugar de la métrica Inception Score (IS) [2], debido a que FID proporciona una evaluación más directa y relevante de la calidad visual y realismo de las imágenes generadas, en comparación con un conjunto de imágenes reales. A diferencia de IS, que se enfoca principalmente en la diversidad y claridad de las imágenes generadas, FID mide cuán similares son estadísticamente las distribuciones de características de las imágenes generadas y un conjunto de imágenes del dataset. Esto es especialmente importante en nuestro trabajo, donde el objetivo es generar imágenes que no solo sean variadas y claras, sino que también sean visualmente indistinguibles de las imágenes auténticas.

En resumen, al incorporar métricas como las comentadas, los evaluadores pueden obtener una comprensión integral del rendimiento de los modelos generativos en la tarea de generación de imágenes. Considerar múltiples métricas nos dará una visión más completa del rendimiento del modelo generativo.

5.3.. Modificaciones

Batch Normalization (BatchNorm): Batch Normalization es una técnica utilizada en redes neuronales para normalizar las activaciones de una capa, aplicándose a lo largo del lote (batch) durante el entrenamiento. Esta normalización ayuda a estabilizar y acelerar el entrenamiento al reducir la dependencia de la inicialización de pesos y mitigar el problema de la desaparición del gradiente. Durante el proceso, se ajustan parámetros de escala y sesgo que permiten a la red aprender la mejor representación para los datos. Esta técnica ya fue bastante utilizada durante la práctica 2 de la asignatura.

Layer Normalization (LayerNorm): Layer Normalization es una técnica similar a Batch Normalization, pero se aplica a lo largo de las características (o canales) de una sola instancia. Es utilizada en tareas no solo de clasificación, sino también en regresión o generación y ajusta parámetros de escala y sesgo.

La principal diferencia entre Batch Normalization y Layer Normalization radica en la dimensión a lo largo de la cual se normalizan las activaciones. BatchNorm opera a lo largo del lote, normalizando las activaciones para todas las instancias en un lote. Por otro lado, LayerNorm normaliza a lo largo de las características para cada instancia individualmente.

5.3.1. Modificación 1

En lugar de LayerNorm se ha utilizado BatchNorm2d. Además, como función de perdida se ha empleado el MSE.

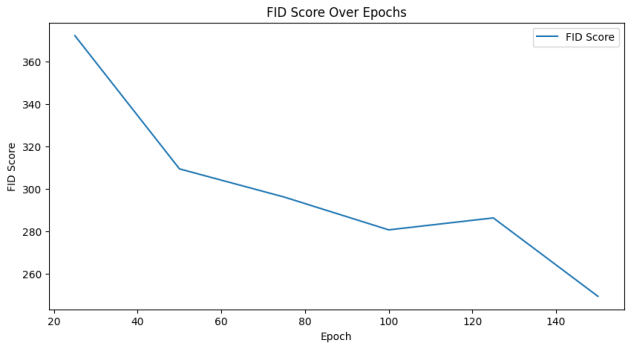


Figura 8. Evolución del FID en la modificación 1

5.3.2. Modificación 2

Se ha utilizado un Batch Size de 128, además de BatchNorm2d en lugar de LayerNorm.

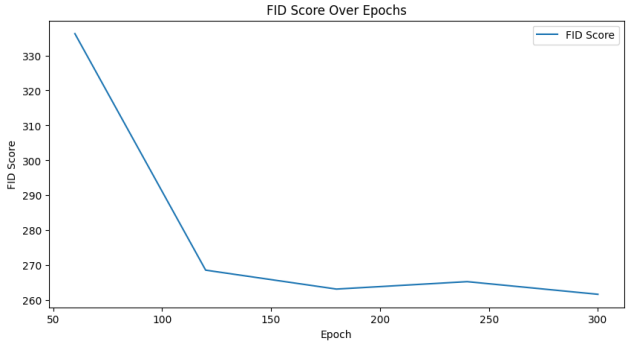


Figura 9. Evolución del FID en la modificación 2

5.3.3. Modificación 3

En este caso, multiplicamos el learning rate por 0.8 cada 25 épocas. Esta estrategia comúnmente conocida como ”decay.º decaimiento del learning rate, permite una convergencia más suave y estable. A medida que el modelo se acerca a un mínimo local o a la solución óptima, reducir el learning rate ayuda a evitar oscilaciones alrededor de ese punto. Esto permite que el modelo .ªjuste finamente”los pesos y mejore la precisión.

En los experimentos realizados se ha obtenido una rápida mejora inicial pero luego no hay apenas aprendizaje.

5.4.. Resultados obtenidos

Para generar imágenes de coches a partir de ruido, como hemos indicado anteriormente, nuestra táctica será revertir el proceso de difusión. Para esto, revertiremos paso a paso hasta llegar al número de ’timesteps’ que hemos establecido.

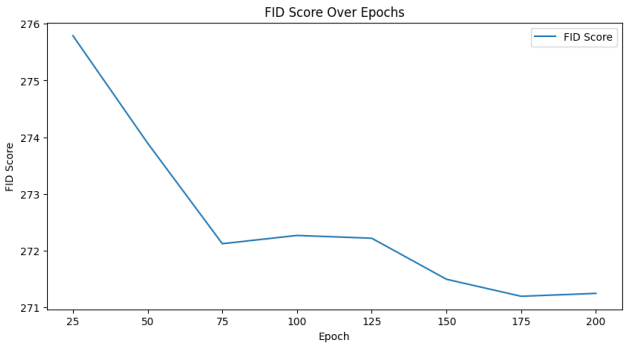


Figura 10. Evolución del FID en la modificación 3

El entrenamiento de nuestro modelo consistirá en que dada una imagen en un paso de ruido t, genere el siguiente paso $(t+1)$, es decir, esa misma imagen pero con menos ruido. Con la función de pérdida, que en la ejecución principal será el MAE, optimizaremos los parámetros del modelo.

Partimos de imágenes de ruido, como vemos a continuación:

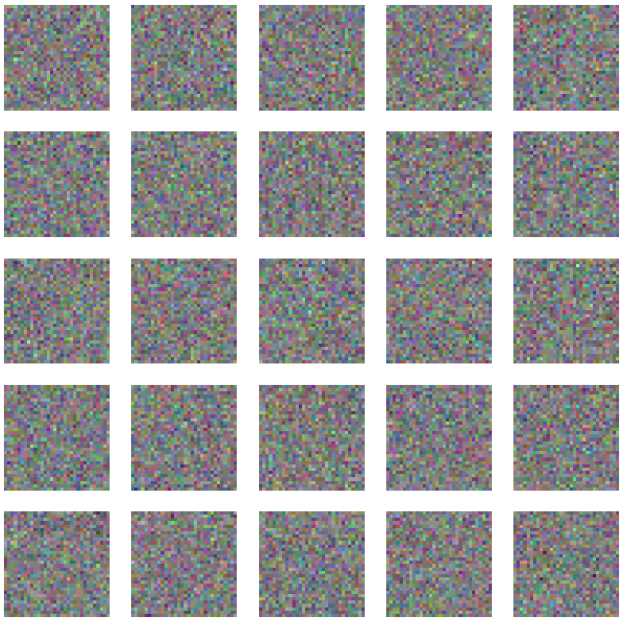


Figura 11. Imágenes iniciales

Finalmente se obtienen las siguientes imágenes de coches:

Tomamos como ejemplo uno de los coches generados. En la siguiente imagen se puede ver el proceso:

Por último, cogemos 5 imágenes fijas y vemos como mejora cada 20 iteraciones:

La ejecución principal ha sido de 400 iteraciones. En las primeras 200 se ha obtenido la siguiente gráfica de FID:

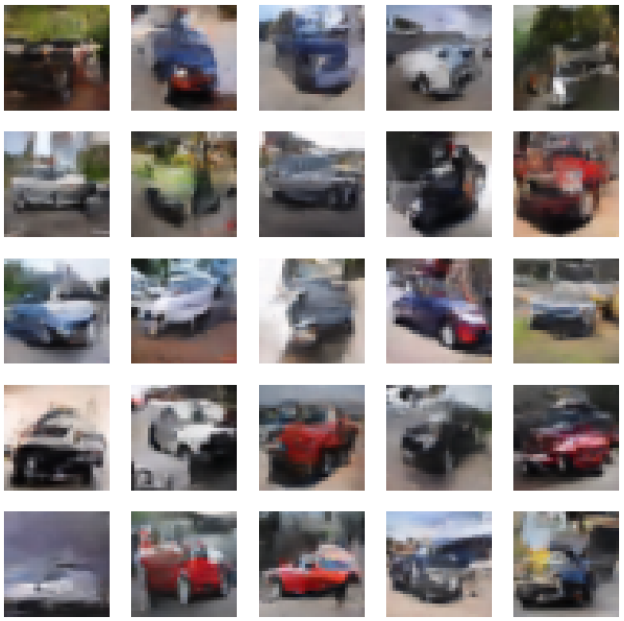


Figura 12. Imágenes finales

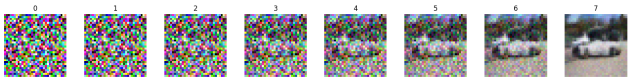
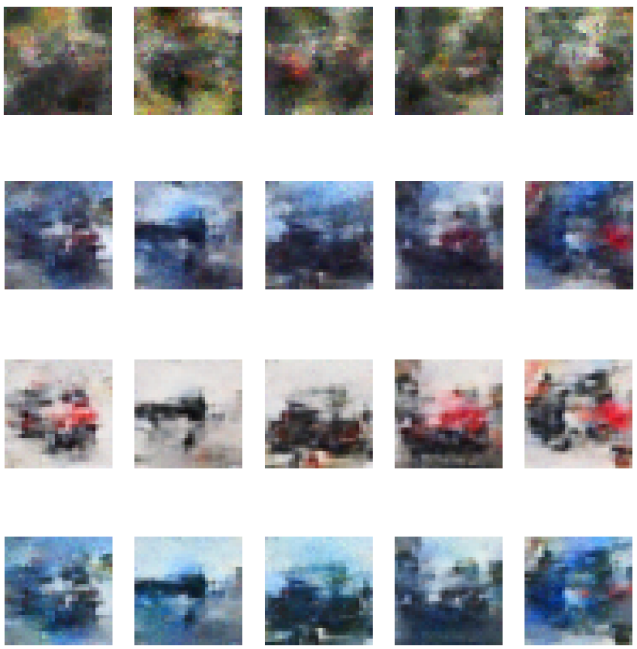
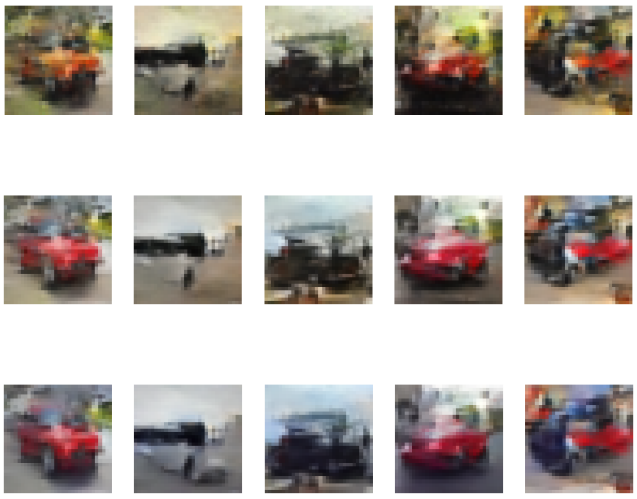


Figura 13. Proceso de creación



Como el tiempo necesario para entrenar era tremendamente grande la ejecución no se ha podido hacer de una vez. En las siguientes 200 iteraciones el FID ha bajado ligeramente hasta llegar a 160.



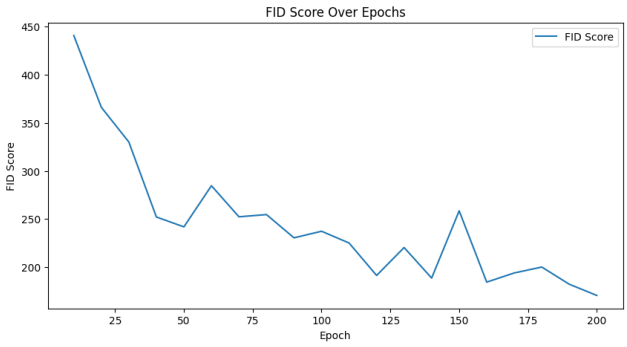


Figura 14. Gráfica de FID de la ejecución principal

6.. Conclusiones

La función de pérdida que mejores resultados ha dado es el MAE. No sólo en la calidad de los resultados, también en cuanto a velocidad de convergencia. Observando las gráficas de FID en las distintas implementaciones, vemos que en los casos en los que se usa el MAE el descenso es mayor.

Por otro lado, no ha habido diferencia entre los experimentos realizados con Batch Normalization y Layer Normalization. También cabe resaltar que las modificaciones introducidas no han llevado a mejoras de resultados. Lo más efectivo ha sido el aumento del número de épocas de entrenamiento. Sin embargo, el tiempo excesivo de entrenamiento nos ha llevado a buscar otras posibles mejoras.

En un posible futuro estudio, se podría tratar de adaptar el cálculo de FID para establecerlo como función pérdida, aunque eso conllevaría un aumento considerable en el tiempo de ejecución.

Referencias

[1] Prafulla Dhariwal Alex Nichol. Improved denoising diffusion probabilistic models. 2021. 1, 2

[2] Shane Barratt and Rishi Sharma. A note on the inception score, 2018. 5

[3] D.C Dowson and B.V Landau. The fréchet distance between multivariate normal distributions. *Journal of Multivariate Analysis*, 12(3):450–455, 1982. 5

[4] Niru Maheswaranathan Surya Ganguli Jascha Sohl-Dickstein, Eric A. Weiss. Deep unsupervised learning using nonequilibrium thermodynamics. 2015. 2

[5] Pieter Abbeel Jonathan Ho, Ajay Jain. Denoising diffusion probabilistic models. 2020. 1, 2

[6] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013. 4

[7] Lilian Weng. What are diffusion models?, 2021. [Online; accessed 14-January-2024]. 2

[8] Wikipedia contributors. Fréchet inception distance — Wikipedia, the free encyclopedia, 2024. [Online; accessed 14-January-2024]. 5