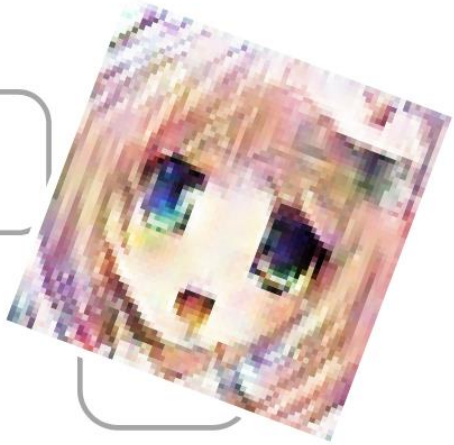




Álvaro Felipe | Miguel Gómez

GANIME: GENERACIÓN DE CARAS DE ANIME CON AUTOENCODERS, VAE Y REDES ADVERSARIAS GENERATIVAS

Métodos Generativos





Índice

Autoencoder

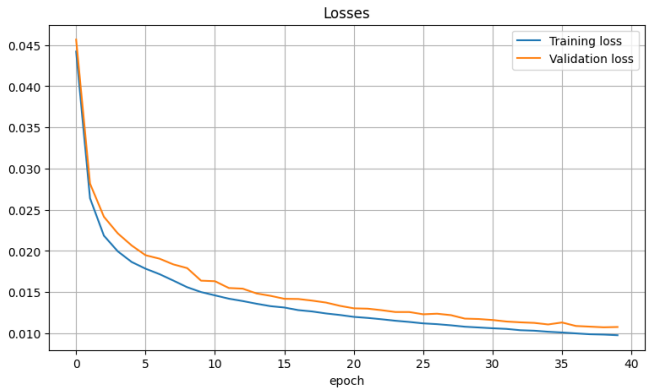
El primer modelo que utilizamos. Como esperábamos, no consiguió buenos resultados, porque no tiene un espacio continuo.

Arquitectura

En la primera versión del Autoencoder (AE), el encoder estaba formado por seis capas convolucionales con un número creciente de filtros, desde 16 hasta 256 (dos capas de 128 filtros). Estas capas tenían filtros de tamaño 3×3, una elección común que ofrece un buen equilibrio entre capacidad de detección de patrones y coste de operaciones.
Tras las capas convolucionales, se incluían dos capas densas de 128 y 64 neuronas, seguidas de una capa de salida densa que representaba el espacio latente de dimensión 32.
El decoder seguía una arquitectura simétrica respecto al encoder, empleando capas Conv2DTranspose para la reconstrucción progresiva de la imagen a partir del espacio latente, y finalizando con una capa convolucional con activación sigmoide que generaba la salida reconstruida.
A lo largo de las versiones sucesivas del modelo se fueron incorporando mejoras. En la versión final, el Autoencoder se simplificó a cinco bloques convolucionales, cada uno compuesto por una capa convolucional, una capa de normalización por lotes (Batch Normalization) y por último les sigue una función de activación ReLU.
El orden de estas operaciones es relevante ya que la normalización se aplica antes de la activación para evitar que la ReLU anulara los valores negativos, provocando una respuesta más lineal.
En esta versión final, el encoder solo cuenta con una capa densa que genera un espacio latente mucho mayor, de 512 dimensiones, permitiendo una codificación más rica. El decoder mantiene una estructura simétrica respecto al encoder y finaliza la reconstrucción con una capa Conv2DTranspose, en lugar de una convolucional viendo que mejora la calidad de las reconstrucciones.

Resultados

Durante el proceso de entrenamiento se observó que el modelo no requería un número elevado de iteraciones para converger. Aproximadamente a partir de las 40 épocas el error de reconstrucción se estabilizaba en valores bajos.
Este comportamiento se debe, en parte, al uso de Batch Normalization, que contribuye a acelerar la convergencia y a mantener la estabilidad del entrenamiento al reducir la dependencia del modelo respecto a la inicialización de los pesos y la escala de las activaciones. El modelo fue entrenado utilizando la función de pérdida MSE, calculada entre los píxeles originales y los píxeles reconstruidos de cada imagen.
Los resultados obtenidos muestran que el Autoencoder logra una compresión efectiva de la información en el espacio latente, preservando las características más relevantes y demostrando que las reconstrucciones son visualmente similares a las originales, aunque presentan un ligero desenfoque, lo cual es esperable debido a la pérdida de detalle inherente al proceso de compresión.

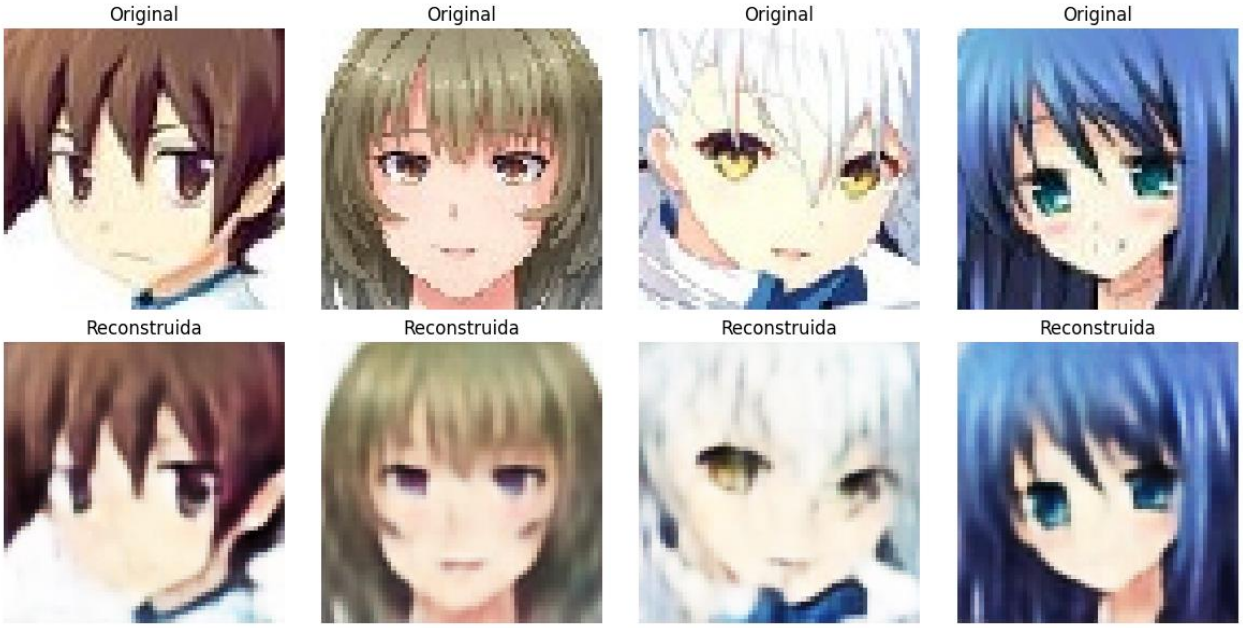


Sin embargo, el modelo presenta dificultades cuando se intenta generar imágenes desde cero a partir de vectores latentes aleatorios de 512 valores. En estos casos, las salidas carecen de

coherencia visual y tienden a parecer ruido estructurado, con solo algunos rasgos parciales reconocibles (como contornos de ojos o formas de cabello típicas de rostros de anime).

Para mitigar este problema, se intentó obtener los vectores latentes promedios correspondientes a un conjunto de imágenes reales (alrededor de 100) y utilizarlos como punto de partida para la generación. No obstante, no se produjeron mejoras significativas, lo que sugiere que el espacio latente no está estructurado como para generar imágenes coherentes de forma directa.

Imágenes reconstruidas:



Generadas:

Autoencoder Variacional

Cogiendo la arquitectura del autoencoder y añadiéndole la capa Sampling, creamos el autoencoder variacional, que es el primer modelo que puede generar imágenes, aunque no estén muy definidas.

Arquitectura

Las arquitecturas que fuimos probando del VAE fueron muy similares en capas al encoder en su versión final, con 5 capas convolucionales con Batch normalization y activación ReLU tanto en el encoder como en el decoder. Las diferentes versiones fueron cambiando el tamaño del espacio latente y la forma de hacer el sampling para que nuestra versión final hayamos implementado la capa de muestreo (Sampling Layer) como una capa personalizada de Keras. Esta capa recibe como entrada z_mean y z_log_var (generadas mediante una capa densa del encoder) y genera un vector latente z aplicando la reparametrización:

$$z = z_{mean} + \exp\left(0.5 \cdot z_{log_{var}}\right) \cdot \epsilon \text{ donde } \epsilon \text{ es ruido normal.}$$

Con esto logramos que la red aprenda distribuciones continuas en el espacio latente necesario para generar imágenes nuevas con el VAE.

El encoder, tras las capas convolucionales aplana la salida y se aplica la capa densa de 512 neuronas para extraer características globales y se calculan z_mean y z_log_var, que se utilizarán en la capa de muestreo. El decoder, simétrico al encoder, comienza con la capa densa de 1024 neuronas cuya función es expandir el vector latente para que pueda reconstruir la estructura espacial, seguida de un

reshape para recuperar las dimensiones, y luego aplica la serie de bloques convolucionales para reconstruir la imagen con una capa Conv2DTranspose de 3 canales y activación sigmoide.

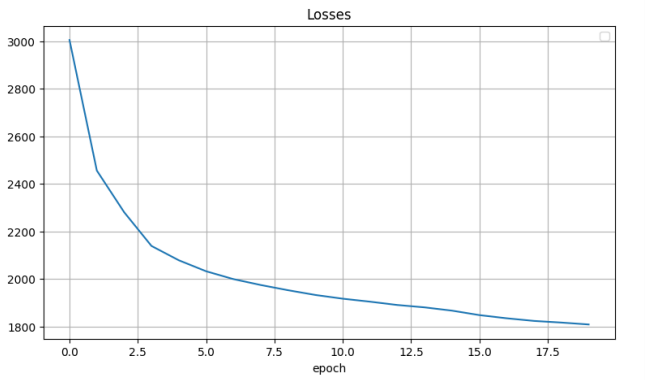
El entrenamiento se realiza mediante una función de pérdida combinada, que suma el error de reconstrucción mediante MAE y la pérdida KL que mide la divergencia entre la distribución latente aprendida y una distribución normal, para garantizar un espacio latente continuo.
El modelo final se crea en una clase VAE (hija de la clase Model de keras), y se define el método train_step para calcular ambas componentes de la pérdida, actualizar los pesos y el cálculo de métrica.

Resultados

La última versión del VAE por fin consigue aprender de forma estable, utilizando como función de pérdida la suma de KL loss y MAE. Esta combinación permite equilibrar la reconstrucción precisa de las imágenes (medida por el MAE) con la regularización del espacio latente (impuesta por la KL). El entrenamiento comienza con un MAE altísimo, lo que indica que las reconstrucciones iniciales son muy pobres. Sin embargo, durante las primeras cinco épocas se observa una mejora significativa: el MAE baja a buen ritmo, lo que sugiere que la red empieza a captar patrones relevantes. A partir de ahí, el descenso se ralentiza y el modelo entra en una fase de estancamiento, donde ya no mejora sustancialmente.

Aunque los datos generados no son especialmente buenos, las caras siguen siendo borrosas y poco detalladas, el modelo sí logra algo que el Autoencoder clásico no podía: generar una cara para cualquier punto del espacio latente. Esto es una mejora importante, ya que implica que el espacio latente aprendido es continuo y estructurado. En otras palabras, el modelo no memoriza ejemplos, sino que aprende una distribución que permite interpolar entre ellos.

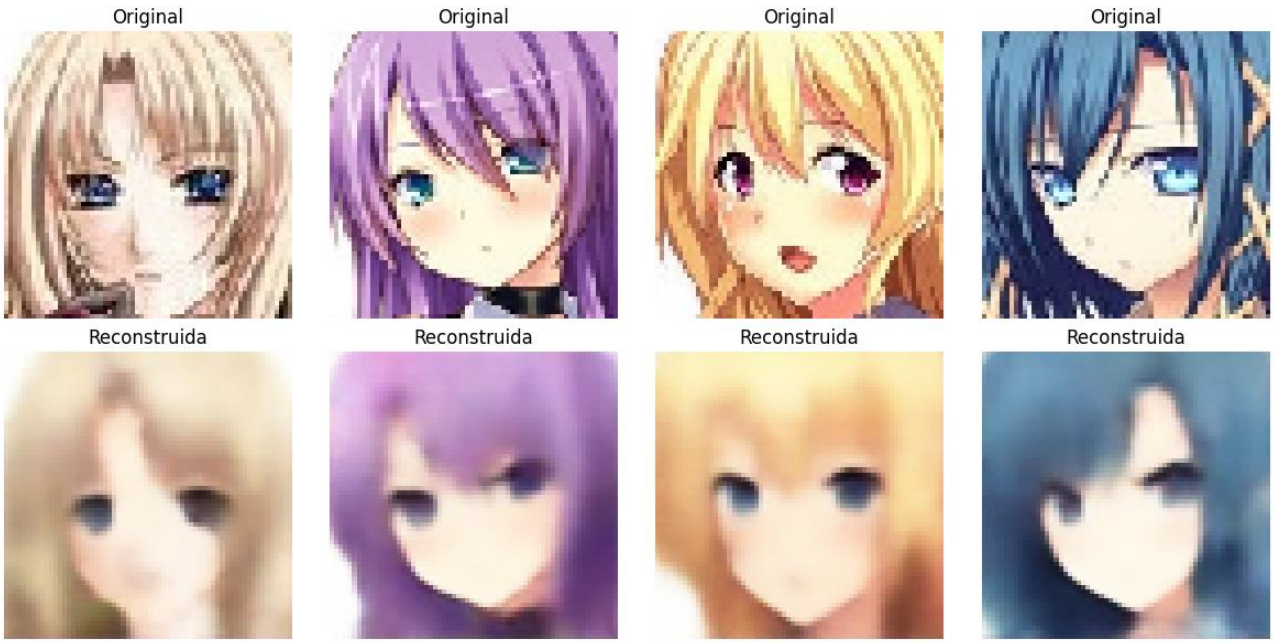
Esto se debe a la pérdida KL, que fuerza al codificador a mapear las entradas en una distribución normal, lo que convierte el espacio latente en un continuo navegable. Gracias a esto, aunque las imágenes no sean perfectas, el modelo puede generar contenido coherente incluso desde vectores aleatorios. Curiosamente, conforme baja la pérdida de reconstrucción, la KL tiende a subir, ya que su magnitud es mucho menor en comparación con el MAE.



Aun así, el modelo no parece perder la continuidad del espacio latente, lo que confirma que la regularización sigue cumpliendo su función.

El resultado es mixto. Por un lado, el modelo cumple con su objetivo teórico: regulariza el espacio latente y permite generación. Por otro, la calidad visual de las imágenes es baja, lo que indica que la capacidad de reconstrucción aún está limitada. Posiblemente se podría mejorar si hubiese estado más tiempo entrenando, ya que nunca ha dejado de mejorar. En cualquier caso, el salto cualitativo respecto al Autoencoder clásico es evidente.

Reconstrucciones:



Generadas:

GAN

Las redes GAN son las que mejor resultados han obtenido con diferencia. No hemos tenido graves problemas de colapso modal, al mantener ambas redes equilibradas.

Para entrenar las redes GAN, hemos seguido 2 enfoques:

- Entrenar la generadora para crear imágenes a color
- Utilizando transfer learning desde la de color y entrenando de cero, entrenarla para generar imágenes en blanco y negro.

El modelo que mejor resultados ha generado ha sido el que genera a color.

Configuración de hiperparámetros
Hemos elegido un learning rate de 2e-4, para intentar hacer el aprendizaje más estable. Tras probar varios tamaños de espacios latentes y viendo que los resultados obtenidos con 100 no son malos, hemos decidido mantenerlo así.

Arquitectura
La arquitectura inicial en ambos enfoques (escala de grises y a color) es la misma, cambiando unicamente el número de canales de salida.

Para entrenar la GAN en blanco y negro, partimos de un *Transfer Learning* desde la a color en todas las capas, menos la última y la entrenamos un poco más. Aun así no obtuvo mejores resultados. Esto se puede deber a que en color capta más detalles y generaliza mejor.

La GAN consiste en 2 redes:

Generador:
El generador parte de un vector latente de tamaño 100, que se genera aleatoriamente, siguiendo una distribución normal.

Primero, el vector se expande mediante una capa densa hasta un tamaño de 4 * 4 * 512, iniciando así el proceso inverso al de una red convolucional tradicional, donde normalmente se parte de imágenes grandes con pocos canales (generalmente 3) y se va reduciendo el tamaño mientras se incrementa el número de canales.

A partir de ahí, se aplican sucesivas capas de convolución transpuesta, con activación ReLU, para ir aumentando la resolución y disminuyendo el número de canales. Cada capa duplica las dimensiones espaciales y reduce a la mitad el número de filtros. La última capa convolucional transpuesta tiene 3 filtros (1 en el caso de imágenes en blanco y negro), y utiliza activación sigmoide, lo que genera la imagen final con valores normalizados entre 0 y 1.

Hemos elegido no aplicar Upsampling, sino todo stride= 2, para que todo se aprenda y no se desprecie información.

Discriminador:
El discriminador recibe como entrada una imagen 64x64. A partir de ahí, aplica una serie de capas convolucionales para extraer características. Empieza con dos capas con 16 y 32 filtros respectivamente, ambas con activación ReLU y padding "same", lo que mantiene las dimensiones espaciales. Luego se aplica una capa de MaxPooling, que reduce la resolución y permite captar patrones más globales.

Después, se añaden tres capas convolucionales con 64, 128 y 256 filtros, también con activación ReLU y padding "same", aumentando progresivamente la profundidad de la red. Una vez extraídas las características, se aplana el tensor y se pasa por dos capas densas de 512 unidades con activación ReLU, que permiten una representación más abstracta. Finalmente, se utiliza una capa densa con una única neurona y activación sigmoide para clasificar la imagen como real o generada.

Resultados

El aprendizaje es bastante estable, con la discriminadora aprendiendo mucho más rápido, ya que, al principio, la generadora solo crea ruido. Cuando la discriminadora sabe diferenciar, la generadora tiene que aprender a hacer imágenes que confundan a la discriminadora.

Así van mejorando ambas, pero llega un momento en el que no parece que mejoran mucho.

Alcanzamos una FID de 127, que es buena, pero se puede mejorar. Por eso probamos haciendo la red más grande.

Con la red generadora en escala de grises, los resultados obtenidos son relativamente similares a los del modelo a color. La reducción a un solo canal ha supuesto una mejora en la velocidad de entrenamiento, pero no ha implicado una mejora en la calidad de los resultados

Mejoras

Para mejorar los resultados hemos intentado múltiples soluciones, que no nos han dado muy buenos resultados:

1. Cambiar las etiquetas de los reales de 1 a 0.9 para ralentizar el aprendizaje de la discriminadora, que era demasiado, pero entonces el discriminador, se quedaba en el 40%, cayendo en colapso modal.
2. Añadir BatchNormalization en el generador, que hemos leído que estabiliza el entrenamiento, pero nunca llega a converger, porque nos aparecía en las imágenes una cuadrícula de color (*checkboard artifacts*).
3. Coger el modelo que nos ha funcionado y duplicar las capas Conv2DTranspose, pero con stride = 1 para mantener el tamaño. Esto no dio más potencia, y nunca llegó a converger.
4. Añadimos también una *skip conection* de la segunda a justo antes de la quinta capa Conv2DTranspose. Generó buenos resultados, pero muy similares a los de nuestra primera

aproximación. A partir de esta decidimos luego probar con *skip connection* para todas las capas.

5. Añadir ruido entre capas, para evitar el colapso modal o los patrones que aparecen con BatchNormalization. No conseguimos generar buenas imágenes con esta capa incluida.
6. Añadir capas residuales y hacer el modelo más profundo para que pueda aprender con mayor calidad. Este modelo, o ejecutamos 4 veces, de las cuales 3 no convergió y la cuarta nos dio nuestro mejor resultado con un FID de 97.
7. Por último, generamos 1000 imágenes y seleccionamos las 200 mejores. Debido a que FID valora más la distribución de las imágenes que la *calidad*, al haber sido generado con los modelos anteriores, no mejoró el FID.

Nuevos resultados:

Al añadir una capa más a las imágenes en color, genera imágenes con menos errores:

Pero al pasar estas imágenes a blanco y negro, salen pixeladas. Se puede ver que estas imágenes generan muy bien los ojos y facciones, pero al mezclar muchos colores, pixela y hace que en escala de grises no se vean bien.

Usando esta nueva arquitectura entrenando directamente en blanco y negro y haciéndole Transfer Learning desde el de color, obtiene mejores resultados.

Consigue un FID de 110, pero no mejora a partir de ahí. Puede deberse a que la arquitectura no da para más o que no hemos tenido suerte en que el entrenamiento se estabilizase.

Por último, las imágenes generadas con la ResGAN, nuestras mejores imágenes:

Viendo que este modelo ha conseguido una mejora sustancial frente a los demás, hubiese sido mejor usar Upsampling + Conv2D en vez de Conv2D Transpose. Y al no usar en esta red casi ninguna capa Conv2D Transpose, la normalización (Batch normalization) es probable que hubiese mejorado también los resultados.