

# Universidad Autónoma de Querétaro

Facultad de Ingeniería  
Maestría en Inteligencia Artificial  
Sheila Leyva López  
Cecilia Gabriela Rodríguez Flores

# Modelos Generativos

07 de junio del 2022

## Resumen

En la actualidad, los avances en inteligencia artificial son cada vez más impresionantes, debido a que el objetivo de las redes neuronales es buscar la manera de imitar el funcionamiento del cerebro humano, sin embargo algo que no habían podido imitar hasta ahora es el tema de la imaginación. Actualmente existe una gran variedad de aplicaciones enfocadas a la creación de información, en donde entra en juego las redes generativas adversariales así como los autocodificadores.

En el presente trabajo se propone la implementación de distintos modelos: Autoencoder(AE), Variational Autoencoder(VAE), Generative Adversarial Network(GAN), Wasserstein GAN with Gradient Penalty (WGAN-GP), Variational Autoencoder with Generative Adversarial Network (VAE-GAN), Generative adversarial interpolative autoencoder(GAIA); para la base de datos CIFAR 10 en el entorno de Google Colaboratory, en donde se presentan los resultados obtenidos con cada uno de los modelos mencionados anteriormente.

## Objetivos

Analizar las diferencias entre seis distintos modelos generativos en Tensorflow 2 utilizando el dataset a elección.

# Especificaciones

Herramientas utilizadas:

- Google Colab Pro
- Tensorflow versión 2.8.2
- Conjunto de datos CIFAR-10 de keras

## Conjunto de datos CIFAR-10

El conjunto de datos CIFAR-10 contiene 60000 imágenes a color en 10 distintas clases, con 6000 imágenes por clases. Específicamente 50000 imágenes están destinadas al entrenamiento y 10000 para pruebas, en la Figura 1 se muestran las clases contenidas..

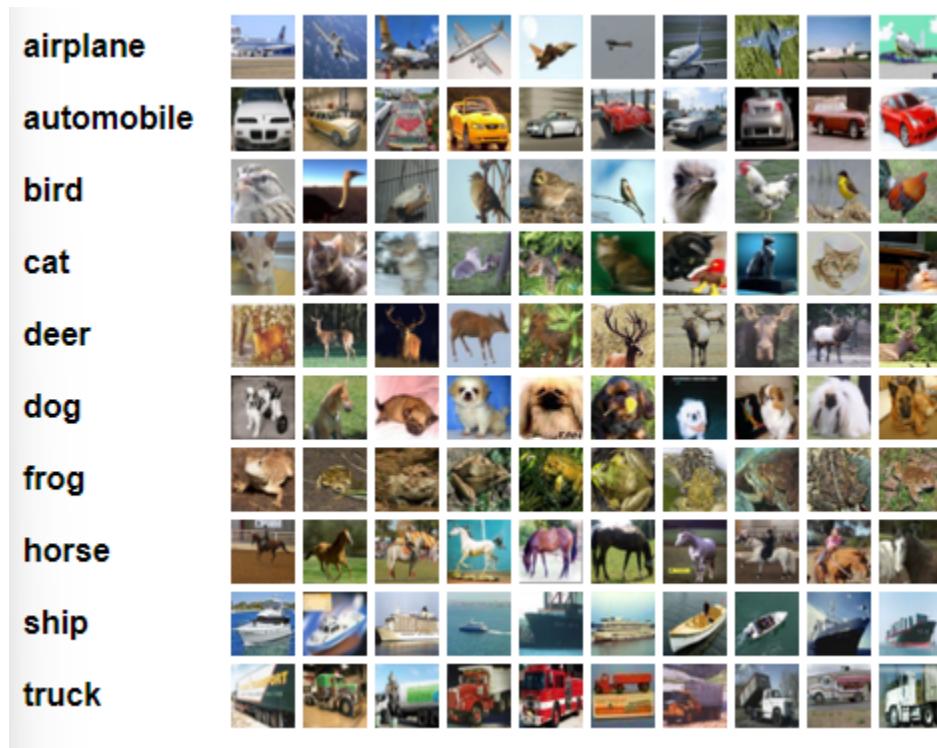


Figura 1. Imágenes de 32x32x3 de 10 distintas clases.

## **Marco Teórico**

Los modelos estadísticos utilizados en el campo de la inteligencia artificial pueden ser generativos o discriminativos. Estos últimos se encargan de aprender a partir de una entrada dada, es decir, aprenden la probabilidad de  $Y$  dado  $X:P(Y|X)$ . Por otro lado, los métodos generativos aprenden la probabilidad conjunta  $P(X|Y)$ .

## **Autoencoder(AE)**

Una de las principales fortalezas de los Autocodificadores (Autoencoders, en inglés), es el aprendizaje de una nueva representación más comprimida de la información. Por tal motivo, el autocodificador se encarga de generar su propia entrada.

Cabe mencionar que la mayoría de los autocodificadores no son considerados modelos generativos. Sin embargo, existe un tipo de autocodificador llamado autocodificador variacional que sí lo es. Sus principales aplicaciones son la compresión de información, detección de fraudes y generación de imágenes.

Existen distintos tipos de autocodificadores, y todos ellos comparten una arquitectura base:

### **Codificador o encoder**

Comprime la información, es decir, se apilan distintas capas neuronales reduciendo en cada una de ellas el número de neuronas.

### **Espacio latente**

Espacio mínimo en el cual la información es representada dentro de la red neuronal.

### **Decodificador o decoder**

Es la actividad inversa del codificador. En cada capa de la red se aumenta el número de neuronas hasta la última capa que tiene el mismo número de neuronas que la primera capa del codificador.

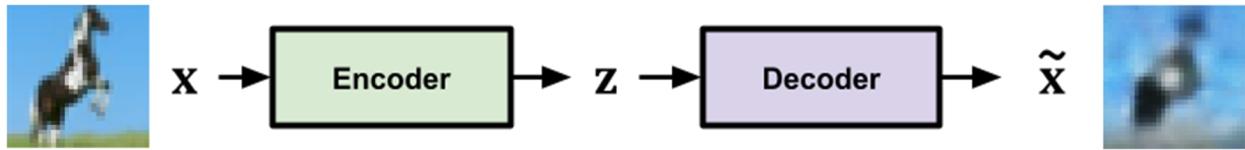


Figura 2. Arquitectura de un autocodificador.

Cada uno de los distintos tipos de autocodificadores se especializa en distintas tareas, como son:

- Autoencoder apilado
- Autocodificador variacional
- Autocodificador disperso
- Denoising Autocodificador

### Variational Autoencoder(VAE)

Un Autocodificador Variacional (VAE) es un autocodificador cuya distribución de codificaciones se regulariza durante el entrenamiento para garantizar que su espacio latente  $z$  cuente con buenas propiedades, con las cuales se puedan generar nuevos datos. Así bien, el término "variacional" proviene de la estrecha relación que existe entre la regularización y el método de inferencia variacional en el área de estadística.

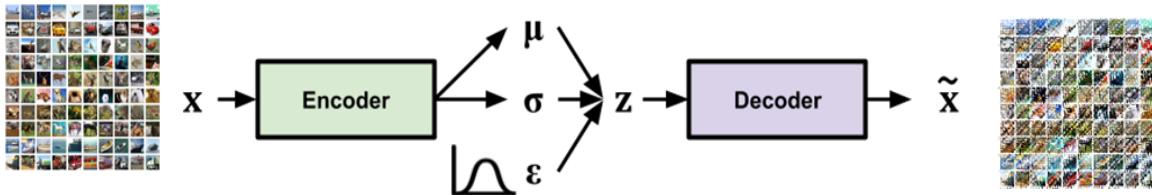


Figura 3. Modelo Autocodificador Variacional (VAE).

Una ventaja a considerar entre los modelos de Autocodificador Variacional (VAE) y los modelos GAN es que pueden asignar una entrada en el conjunto de datos original a factores latentes y luego a una imagen en la aproximación del generador. Sin embargo, las imágenes de muestra

generadas por estos modelos suelen ser borrosas y de menor calidad en comparación con las imágenes resultantes de los modelos GAN.

## Generative Adversarial Network(GAN)

Las Redes Generativas Antagónicas o Adversarias (GAN, por sus siglas en inglés) fueron propuestas por Ian Goodfellow en el 2014. En una GAN, básicamente, se constituye de dos modelos de redes neuronales profundas, los cuales toman el papel de generador y discriminador, con el fin de competir entre sí.

- **Discriminador:** debe decir si la imagen es real o falsa.
- **Generador:** debe crear las imágenes de tal forma que se asemejen a las imágenes auténticas.

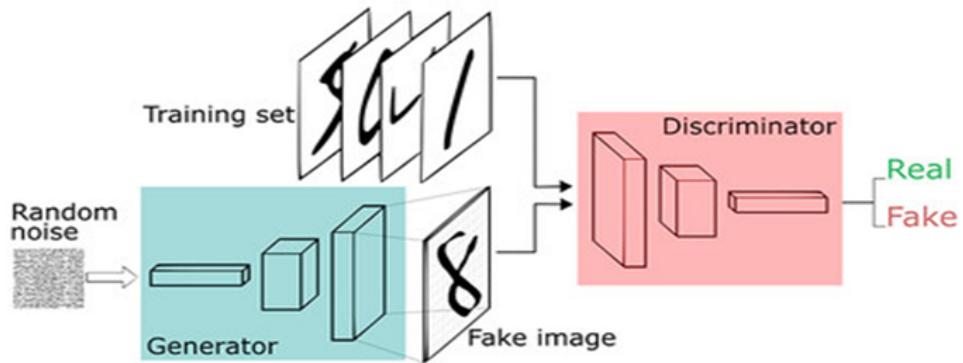


Figura 4. Modelo Generativo Adversarial (GAN).

Por consiguiente, estas redes son consideradas adversarias, debido a que realizan tareas contrarias, es decir, terminan participando en un juego de suma cero, en donde cuando una red gana, la otra pierde. Una de las aplicaciones más comunes y por las cuales surgen las GANs es la generación de imágenes, ya sea de rostros, dígitos, lugares, entre muchos otros.

Para el entrenamiento de las GANs se requiere de un conjunto de datos reales. En un inicio resulta muy fácil para el discriminador acertar cuando determina si las fotos son reales o no, sin

embargo, a medida que el generador mejora sus resultados, la tarea del discriminador se vuelve más complicada de realizar.

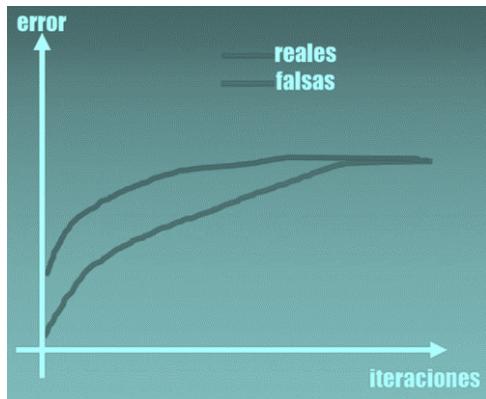


Figura 5. A mayor número de iteraciones es más difícil para el discriminador notar la diferencia entre resultados reales o falsos.

Desde la introducción del concepto del modelo GAN se han desarrollado distintas versiones para intentar mejorar el desempeño original de las GANs. Ejemplo de ello son las redes neuronales generativas adversarias progresivas: básicamente, se aumenta la resolución de las imágenes progresivamente, cuando los resultados ya son lo suficientemente buenos, es decir, si en la salida se tienen imágenes funcionales de 4x4 píxeles, se aumenta a 8x8 píxeles, luego a 16x16 píxeles, así sucesivamente hasta que la calidad de la imagen sea la requerida.



Figura 6. Modelo GANs progresivos.

Otra alternativa, es influenciar las imágenes que arroja el generador, a través de una condición dada al generador y discriminador. Donde dicha condición pueden ser bocetos, imágenes de lugares, objetos e incluso texto.

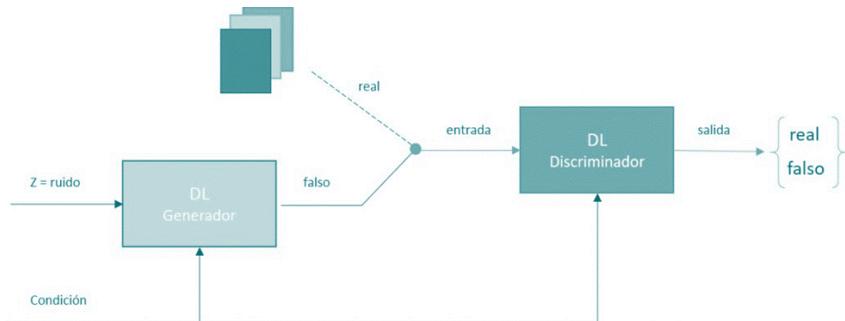


Figura 7. Modelo GANs condicionadas-

### **Wasserstein GAN with Gradient Penalty (WGAN-GP)**

La Wasserstein GAN o WGAN fue presentada por Martin Arjovsky, et al. en su artículo de 2017 titulado “ **Wasserstein GAN** ”. Es una extensión de las GANs debido a que busca una forma alternativa de entrenar el modelo generador para aproximarse mejor a la distribución de datos observados en un conjunto de datos de entrenamiento determinado.

En lugar de usar un discriminador para clasificar o predecir la probabilidad de que las imágenes generadas sean reales o falsas, la WGAN cambia o reemplaza el modelo discriminador con un crítico que califica la realidad o la falsedad de una imagen determinada.

Este cambio está motivado por un argumento teórico de que el entrenamiento del generador debe buscar una minimización de la distancia entre la distribución de los datos observados en el conjunto de datos de entrenamiento y la distribución observada en los ejemplos generados.

El beneficio de WGAN es que el proceso de entrenamiento es más estable y menos sensible a la arquitectura del modelo y la elección de configuraciones de hiperparámetros. Quizás lo más

importante es que la pérdida del discriminador parece estar relacionada con la calidad de las imágenes creadas por el generador.

La WGAN propuesta avanza hacia el entrenamiento estable de GAN, pero a veces aún puede generar solo muestras de baja calidad o no converger. Estos problemas a menudo se deben al uso de recorte de peso en WGAN para imponer una restricción de Lipschitz en el crítico, lo que puede conducir a un comportamiento no deseado.

Por lo tanto, los creadores del modelo WGAN-GP propusieron una alternativa al recorte de pesos: penalizar la norma de gradiente de la crítica respecto a su entrada, con lo cual el método propuesto funciona mejor que la WGAN estándar y permite el entrenamiento estable de una amplia variedad de arquitecturas GAN casi sin ajuste de hiperparámetros, incluidos el modelo ResNet de 101 capas y los modelos de lenguaje sobre datos discretos.

Así bien, una penalización de gradiente es una versión suave de la restricción de Lipschitz, que se deriva del hecho de que las funciones son 1-Lipschitz si y sólo si los gradientes son de norma como máximo 1 en todas partes. La diferencia al cuadrado de la norma 1 se utiliza como penalización de gradiente.

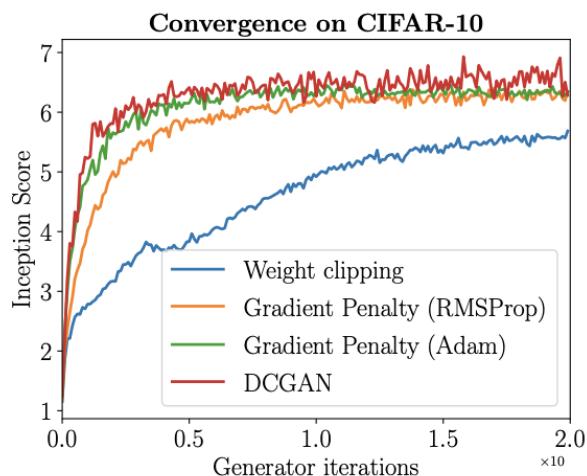


Figura 8. Rendimiento WGAN con penalización del gradiente.

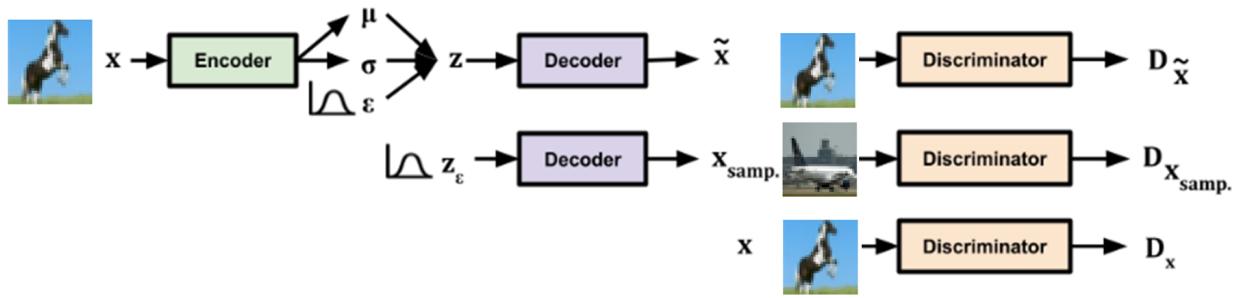


Figura 9. Modelo WGAN.

### Variational Autoencoder with Generative Adversarial Network (VAE-GAN)

El término VAE-GAN se introdujo por primera vez en el artículo "Autoencoding beyond pixels using a learned similarity metric" de A. Larsen et. al., en donde sugieren que la combinación de autocodificadores variacionales y redes generativas adversariales supera a los autocodificadores variacionales tradicionales.

Mientras que un autocodificador variacional aprende a codificar la entrada dada como puede ser el caso de una imagen, para posteriormente reconstruirla a partir de la codificación, una red generativa adversarial (GAN) trabaja para la generación de nuevos datos, los cuales no pueden distinguirse de los datos reales. La clave de su funcionamiento es que, en el caso de un autocodificador, se utilizan las representaciones latentes generadas por un codificador para diversas tareas.

Para obtener los beneficios de ambos, el trabajo (Larsen et al., 2015) propuso el modelo VAE-GAN, el cual añade un discriminador sobre la imagen generada. La función de pérdida para el discriminador es la misma que la del modelo GAN. La pérdida para el decodificador y el codificador tiene dos componentes: La primera componente viene de la misma ecuación del modelo VAE, mientras que el segundo componente se minimiza cuando el generador consigue engañar al discriminador.

Por tal motivo, el modelo VAE-GAN genera con éxito imágenes de estilo GAN mientras preserva la funcionalidad para mapear una imagen de muestra de vuelta a sus variables latentes.

## Generative adversarial interpolative autoencoder (GAIA)

El Autoencoder Generativo Adversarial Interpolativo (GAIA) es un nuevo híbrido entre la Red Generativa Adversarial (GAN) y el Autoencoder (AE). El propósito de GAIA es superar tres problemas que existen en las GAN y los AE:

### 1. Las GAN no son bidireccionales

Mientras que los autocodificadores son capaces de traducir del dominio de la imagen ( $X$ ) al espacio latente de una red neuronal ( $Z; X \rightarrow Z$ ) a través del codificador, y del espacio latente de vuelta al dominio de la imagen a través del decodificador ( $Z \rightarrow X$ ), las GAN sólo tienen un decodificador (generador), y sólo pueden traducir de  $Z \rightarrow X$

### 2. Los autocodificadores producen imágenes borrosas

Los autocodificadores se entranan con funciones de error a nivel de píxel. Como resultado, cuando hay contrastes marcados en las imágenes y éstas se comprimen, los autocodificadores suavizan esos contrastes, produciendo imágenes que no parecen muy realistas.

### 3. Los espacios latentes del autoencoder no son convexas

Los espacios latentes generativos presentan la poderosa propiedad de poder interpolar suavemente entre señales del mundo real en un espacio de alta dimensión mediante la interpolación lineal entre sus representaciones en el espacio latente. Las interpolaciones en un espacio latente de baja dimensión suelen producir representaciones comprensibles cuando se proyectan de nuevo en un espacio de alta dimensión. Sin embargo, en los espacios latentes de muchas arquitecturas de red (como los EA) las interpolaciones lineales no están necesariamente justificadas, porque las proyecciones del espacio latente no se entranan explícitamente para formar un conjunto convexo.

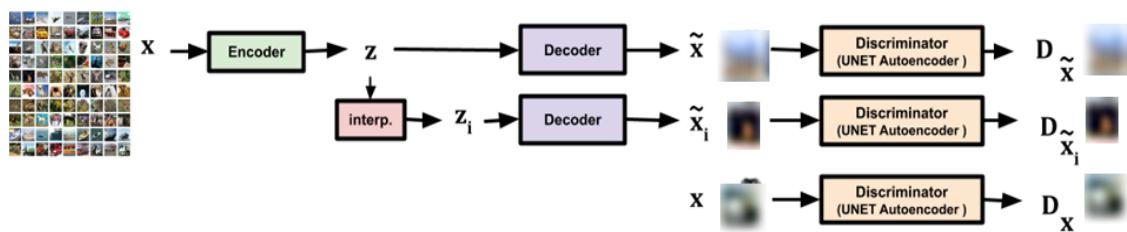


Figura 10. Modelo Autocodificador Generativo Adversarial Interpolativo (GAIA).

## Desarrollo

Para la elaboración de este trabajo, se hizo uso de los modelos Autoencoder(AE), Variational Autoencoder(VAE), Generative Adversarial Network(GAN), Wasserstein GAN with Gradient Penalty (WGAN-GP), Variational Autoencoder with Generative Adversarial Network (VAE-GAN), Generative adversarial interpolative autoencoder(GAIA). La metodología utilizada para la implementación de los modelos mencionados anteriormente fue muy similar entre sí.

Así bien, fue necesario incorporar a la mayoría de los modelos las siguientes líneas mostradas en la figura 11, debido a que se contó con ciertos inconvenientes al querer ejecutar los códigos propuestos del repositorio de ReposHub.

```
[30] import tensorflow.compat.v1.keras.backend as K  
import tensorflow as tf  
tf.compat.v1.enable_eager_execution()  
  
[31] tf.compat.v1.disable_tensor_equality()
```

Figura 11. Líneas añadidas para la ejecución de los modelos propuestos.

Las líneas de código mostradas permiten ejecutar funciones de Tensorflow, tanto versiones anteriores como actuales, lo que elimina problemas de incompatibilidad.

Antes de proseguir con los resultados obtenidos de cada uno de los modelos propuestos, se procede a visualizar en la siguiente tabla los parámetros asociados a cada uno de los modelos propuestos.

TABLA 1  
Parámetros asociados a los modelos propuestos.

| Modelos propuestos  | TRAIN<br>BUF | BATCH<br>SIZE | TEST<br>BUF | DIMS        | Opt  | Loss |
|---------------------|--------------|---------------|-------------|-------------|------|------|
| Autoencoder<br>(AE) | 60000        | 512           | 10000       | (32, 32, 3) | Adam | MSE  |

|                                                                              |       |     |       |             |                 |                                                                                                                          |
|------------------------------------------------------------------------------|-------|-----|-------|-------------|-----------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>Variational Autoencoder (VAE)</b>                                         | 60000 | 512 | 10000 | (32, 32, 3) | Adam            | recon_loss<br>latent_loss                                                                                                |
| <b>Generative Adversarial Network (GAN)</b>                                  | 60000 | 512 | 10000 | (32, 32, 3) | Adam<br>RMSprop | disc_loss<br>gen_loss                                                                                                    |
| <b>Wasserstein GAN with Gradient Penalty (WGAN-GP)</b>                       | 60000 | 512 | 10000 | (32, 32, 3) | Adam<br>RMSprop | disc_loss<br>gen_loss                                                                                                    |
| <b>Variational Autoencoder with Generative Adversarial Network (VAE-GAN)</b> | 60000 | 64  | 10000 | (32,32,3)   | Adam<br>RMSprop | 'd_prop',<br>'latent_loss',<br>'discrim_layer_recon_loss',<br>'gen_fake_loss',<br>'disc_fake_loss',<br>'disc_real_loss', |
| <b>Generative adversarial interpolative autoencoder (GAIA)</b>               | 60000 | 64  | 10000 | (32, 32,3)  | Adam<br>RMSprop | d_xg_loss<br>d_xi_loss<br>d_x_loss<br>xg_loss                                                                            |

## Resultados obtenidos

Al realizar la ejecución de cada uno de los modelos fue posible obtener diversos resultados para cada uno de los modelos propuestos, por lo cual a continuación se mostrarán los resultados obtenidos por medio de tablas y figuras.

## Autoencoder

En las Figuras 11, 12 y 13, se muestra el resultado de haber codificado y decodificado imágenes del conjunto de datos CIFAR-10, en 50, 100 y 150 épocas, respectivamente. Asimismo, en la Tabla 2 se muestra el resumen de Medias de Errores Cuadráticos para cada época.

TABLA 2  
Métricas del modelo asociadas al número de épocas.

| Épocas | MSE      |
|--------|----------|
| 50     | 0.007779 |
| 100    | 0.007485 |
| 150    | 0.007767 |

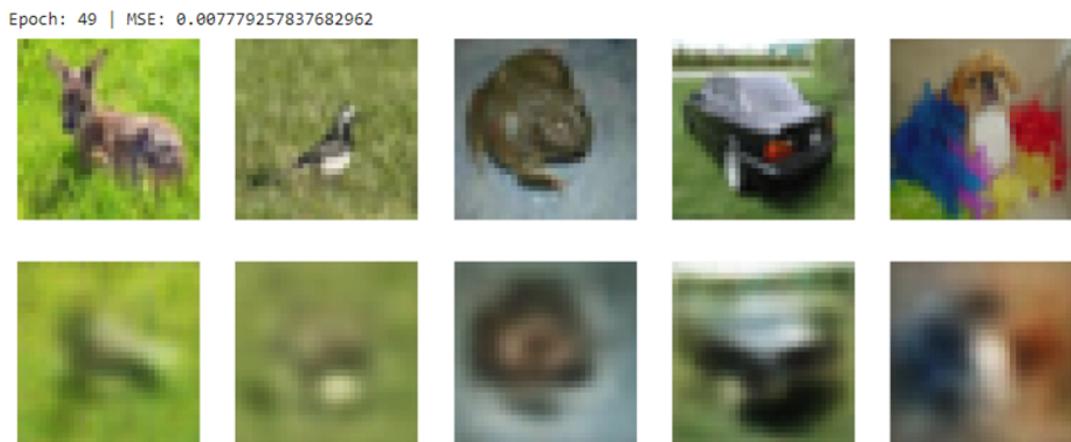


Figura 12. Resultados obtenidos con el modelo AE con 50 épocas.

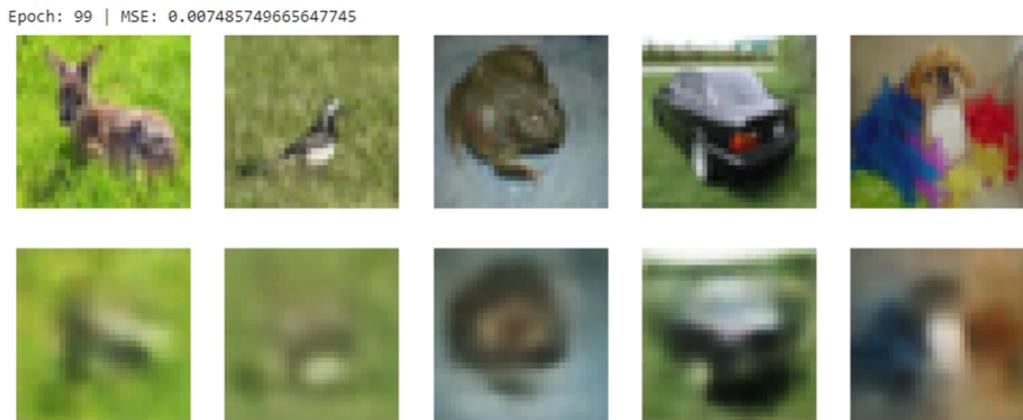


Figura 13. Resultados obtenidos con el modelo AE con 100 épocas.

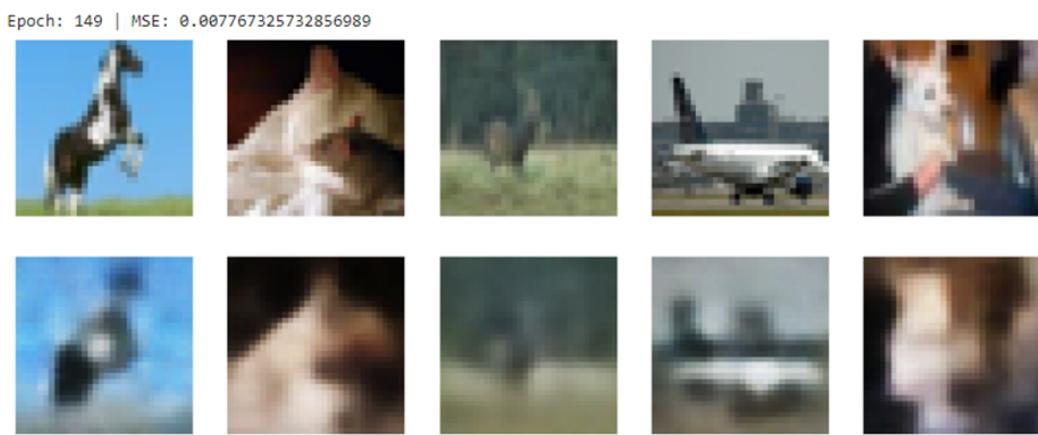


Figura 14. Resultados obtenidos con el modelo AE con 150 épocas.

### Variational - Autoencoder (VAE)

En las Figuras 15-18, se muestra el resultado de haber utilizado el Autocodificador Variacional con imágenes del conjunto de datos CIFAR-10, en 5, 10, 50 y 100 épocas. Asimismo, en la Tabla 3 se muestra el resumen de funciones de pérdida para la reconstrucción y latencia.

TABLA 3  
RESULTADOS OBTENIDOS MODELO VAE

| Epochs | recon_loss         | laten_loss         |
|--------|--------------------|--------------------|
| 5      | 13.821849822998047 | 2.398463249206543  |
| 10     | 13.107399940490723 | 2.3641421794891357 |
| 50     | 13.44339656829834  | 2.4038374423980713 |
| 100    | 13.080192565917969 | 2.4204814434051514 |

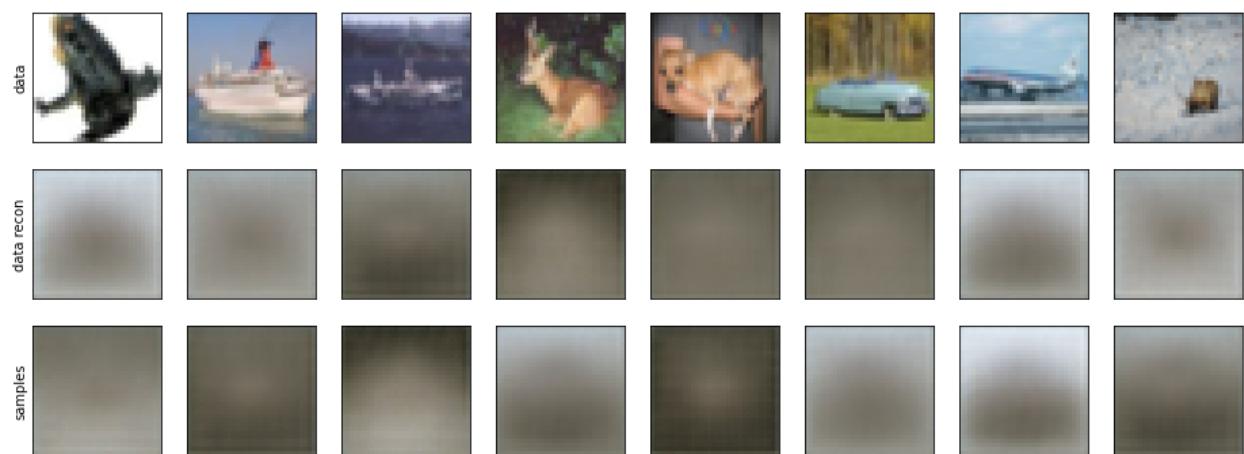


Figura 15. Resultados obtenidos con el modelo VAE con 5 épocas.

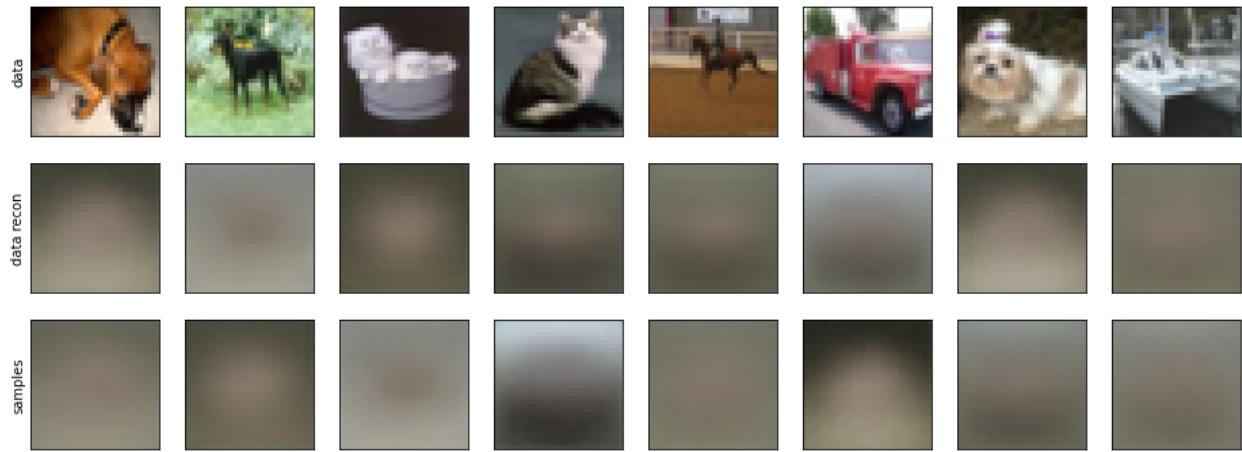


Figura 16. Resultados obtenidos con el modelo VAE con 10 épocas.

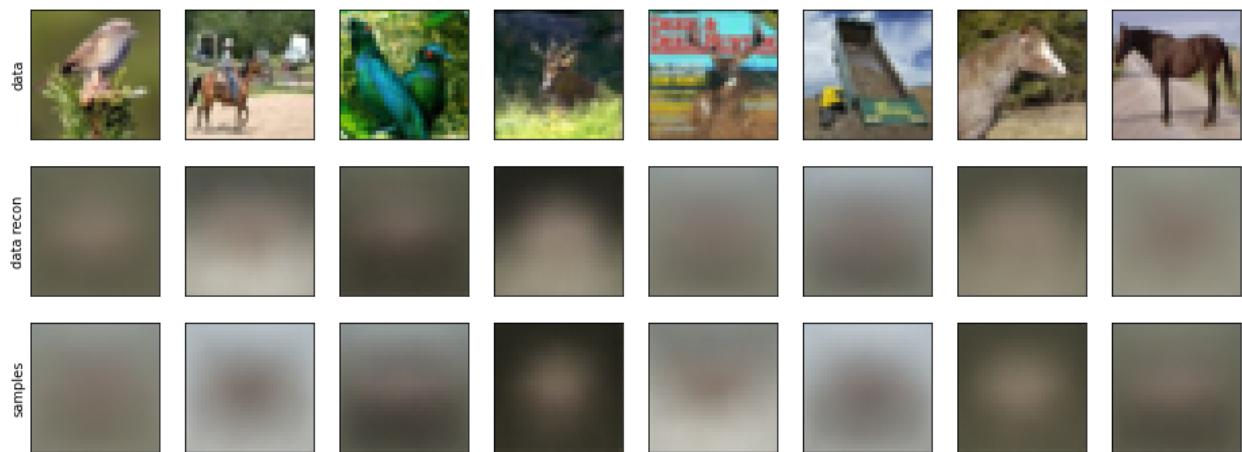


Figura 17. Resultados obtenidos con el modelo VAE con 50 épocas.

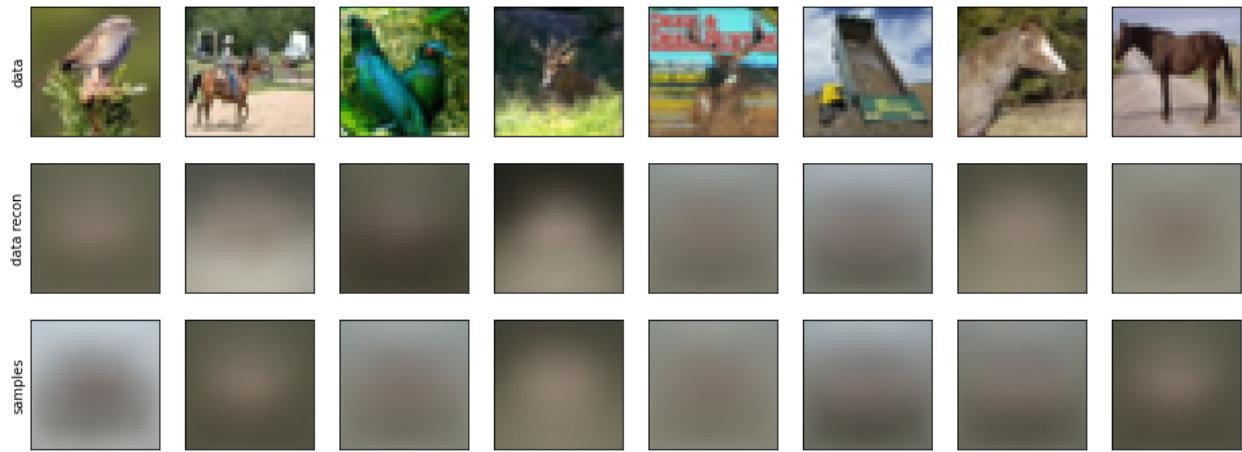


Figura 18. Resultados obtenidos con el modelo VAE con 100 épocas.

## GAN

En las Figuras 19,20 y 21, se muestra el resultado de haber utilizado la Red Generativa Adversarial con imágenes del conjunto de datos CIFAR-10 para 50 épocas. Para cada época se variaron las tasas de aprendizaje del optimizador, tanto del generador como del discriminador. Asimismo, en la Tabla 4 se muestra el resumen de funciones de pérdida para el discriminador y generador.

TABLA 4  
Resultados obtenidos del modelo GAN con 50 épocas

| Modelos generativos                 | Gen_optimizer<br>(Adam) | Disc_optimizer<br>(RMSprop) | Pérdida discriminador | Pérdida generador  |
|-------------------------------------|-------------------------|-----------------------------|-----------------------|--------------------|
| Generative Adversarial Network(GAN) | 0.001                   | 0.005                       | 1.3866125345230103    | 0.7030718922615051 |
|                                     | 0.01                    | 0.05                        | 9.96210780562128e-15  | 32.2401237487793   |
|                                     | 0.0001                  | 0.0005                      | 9.96580341037015e-15  | 32.239994049072266 |

Epoch: 49 | disc\_loss: 1.3866125345230103 | gen\_loss: 0.7030718922615051



Figura 19. Resultados obtenidos con el modelo GAN con 50 épocas, gen\_optimizer=0.001 y disc\_optimizer = 0.005.

Epoch: 49 | disc\_loss: 9.96210780562128e-15 | gen\_loss: 32.2401237487793

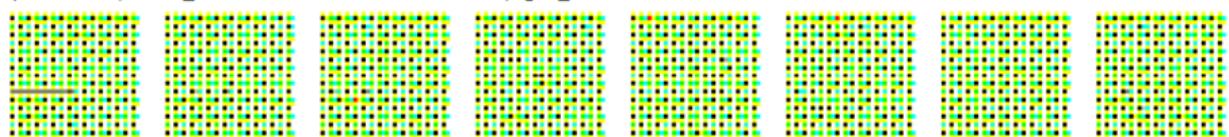


Figura 20. Resultados obtenidos con el modelo GAN con 50 épocas, gen\_optimizer=0.01 y disc\_optimizer = 0.05.

Epoch: 49 | disc\_loss: 9.96580341037015e-15 | gen\_loss: 32.239994049072266

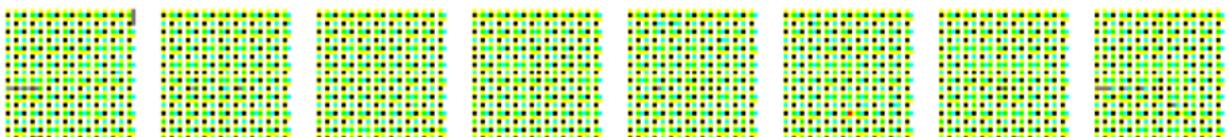


Figura 21. Resultados obtenidos con el modelo GAN con 50 épocas, gen\_optimizer=0.0001 y disc\_optimizer = 0.0005.

En las Figuras 22, 23 y 24, se muestra el resultado de haber utilizado la Red Generativa Advesarial con imágenes del conjunto de datos CIFAR-10 para 100 épocas. Para cada época se variaron las tasas de aprendizaje del optimizador, tanto del generador como del discriminador. Asimismo, en la Tabla 5 se muestra el resumen de funciones de pérdida para el discriminador y generador.

TABLA 5  
Resultados obtenidos del modelo GAN con 100 épocas

| Modelos generativos                 | Gen_optimizer | Disc_optimizer | Pérdida discriminador | Pérdida generador  |
|-------------------------------------|---------------|----------------|-----------------------|--------------------|
| Generative Adversarial Network(GAN) | 0.001         | 0.005          | 1.3862947225570679    | 0.6931473612785339 |
|                                     | 0.01          | 0.05           | 1.3864511251449585    | 0.7057249546051025 |
|                                     | 0.0001        | 0.0005         | 1.1299501657485962    | 0.8322003483772278 |

Epoch: 99 | disc\_loss: 1.3862947225570679 | gen\_loss: 0.6931473612785339

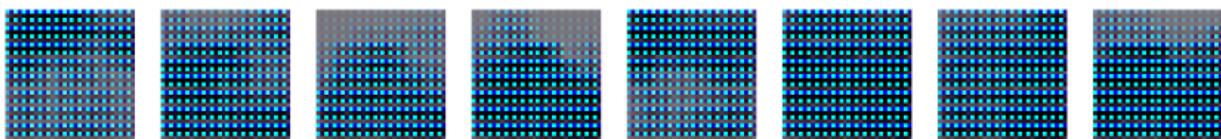


Figura 22. Resultados obtenidos con el modelo GAN con 100 épocas, gen\_optimizer=0.001 y disc\_optimizer = 0.005.

Epoch: 49 | disc\_loss: 1.3864511251449585 | gen\_loss: 0.7057249546051025

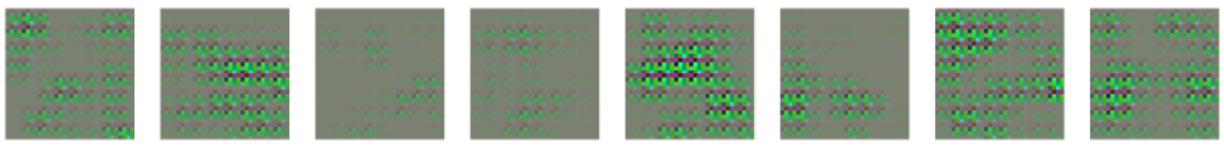


Figura 23. Resultados obtenidos con el modelo GAN con 100 épocas, gen\_optimizer=0.01 y disc\_optimizer = 0.05.

Epoch: 99 | disc\_loss: 1.1299501657485962 | gen\_loss: 0.8322003483772278



Figura 24. Resultados obtenidos con el modelo GAN con 100 épocas, gen\_optimizer=0.0001 y disc\_optimizer = 0.0005.

## **Wasserstein GAN with Gradient Penalty (WGAN-GP)**

En la Figura 25, se muestra el resultado de haber utilizado las WGAN-GP con imágenes del conjunto de datos CIFAR-10. Para cada época se utilizaron las mismas tasas de aprendizaje de los optimizadores, tanto del generador como del discriminador. Asimismo, en la Tabla 6 se muestra el resumen de funciones de pérdida para el discriminador y generador.

TABLA 6  
Resultados obtenidos con el modelo WGAN-GP

| Modelos generativos                             | Épocas | Gen_optimizer | Disc_optimizer | Pérdida discriminador | Pérdida generador   |
|-------------------------------------------------|--------|---------------|----------------|-----------------------|---------------------|
| Wasserstein GAN with Gradient Penalty (WGAN-GP) | 10     | 0.0001        | 0.0005         | 4.780920505523682     | 0.07378218322992325 |
|                                                 | 20     | 0.0001        | 0.0005         | nan                   | nan                 |
|                                                 | 100    | 0.0001        | 0.0005         | nan                   | nan                 |

Las épocas 20 y 100 no fueron completadas por cuestiones limitantes de software y hardware. Por tal motivo no aparecen valores para las funciones de pérdida del discriminador y generador.

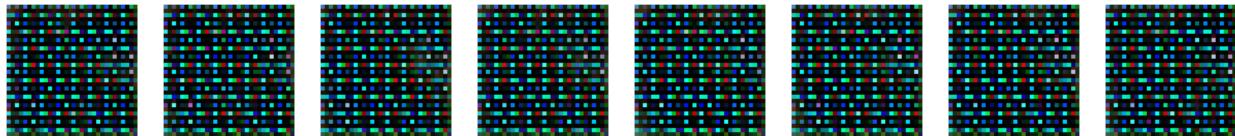


Figura 25. Resultados obtenidos con el modelo WGAN-GP con 10 épocas.

En la Figura 26 y 27 se muestra el historial de los valores de la función de pérdida para 10 épocas, tanto del generador como del discriminador.

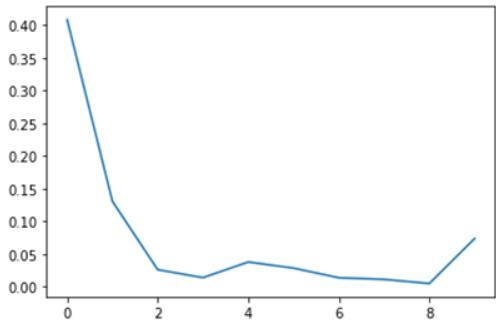


Figura 26. Gráfica obtenida de los valores de pérdida del generador del Modelo WGAN-GP con 10 épocas.

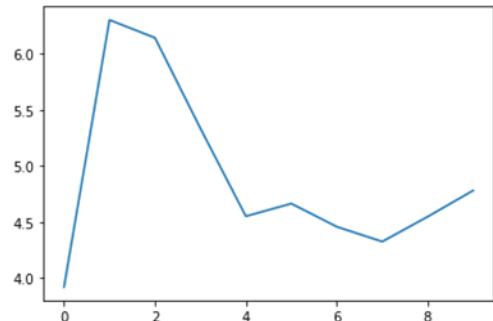


Figura 27. Gráfica obtenida de los valores de pérdida del discriminador del Modelo WGAN-GP con 10 épocas.



Figura 28. Visualización de los resultados obtenidos con el modelo WGAN-GP con 20 épocas.

En la Figura 28 y 29 se muestra el historial de los valores de la función de pérdida para 20 épocas, tanto del generador como del discriminador.

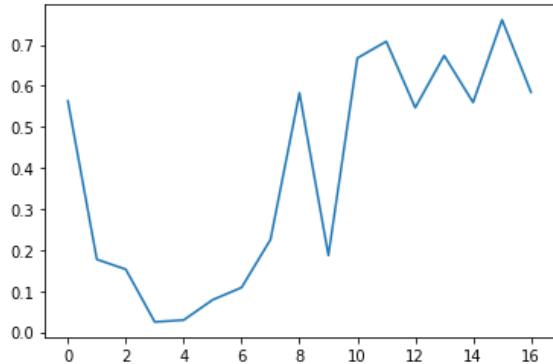


Figura 29. Gráfica obtenida de los valores de pérdida del generador del Modelo WGAN-GAN con 20 épocas.

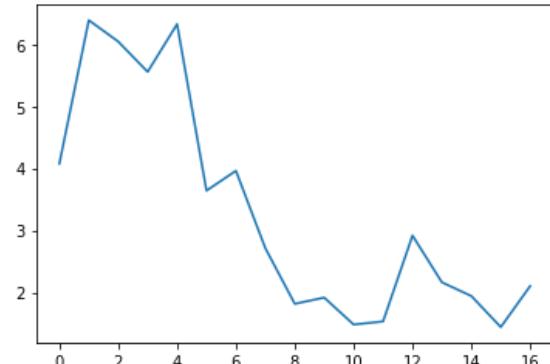


Figura 30. Gráfica obtenida de los valores de pérdida del discriminador del Modelo WGAN-GAN con 20 épocas.

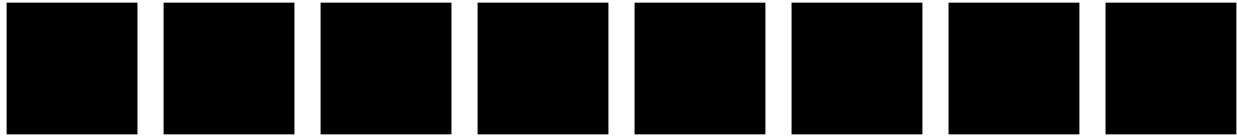


Figura 31. Visualización de los resultados obtenidos con el modelo WGAN-GP con 100 épocas.

En la Figura 32 y 33 se muestra el historial de los valores de la función de pérdida para 100 épocas, tanto del generador como del discriminador.

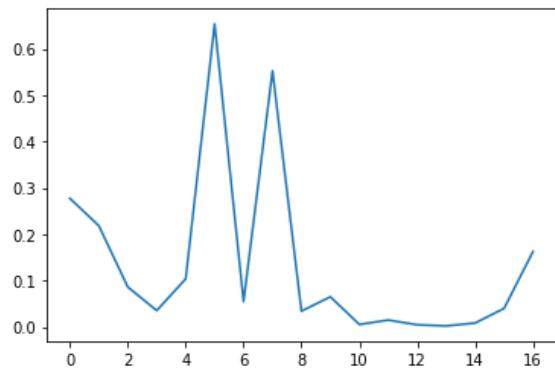


Figura 32. Gráfica obtenida de los valores de pérdida del generador del Modelo WGAN-GP con 100 épocas.

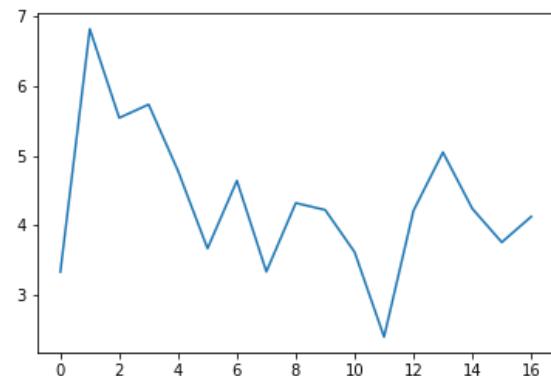


Figura 33. Gráfica obtenida de los valores de pérdida del discriminador del Modelo WGAN-GP con 100 épocas.

## Variational - Autoencoder with Generative Adversarial Network (VAE-GAN)

En las Figuras 34,36 y 38 se muestra el resultado de haber utilizado el modelo VAE-GAN con imágenes del conjunto de datos CIFAR-10. Asimismo en la Tabla 7 se muestran, para cada época (10, 50 y 100), el resumen de funciones de pérdida para el discriminador y generador, dichos gráficos se muestran en las Figuras 35, 37 y 39.

TABLA 7  
RESULTADOS OBTENIDOS DEL MODELO VAE-GAN

| Épocas | gen_fake_loss      | disc_fake_loss      | disc_real_loss      |
|--------|--------------------|---------------------|---------------------|
| 10     | 2.0443601608276367 | 0.20735062658786774 | 0.48760202527046204 |
| 50     | 1.581476092338562  | 0.6593310236930847  | 0.11215417832136154 |
| 100    | 2.602600574493408  | 0.3464422821998596  | 0.10107631236314774 |

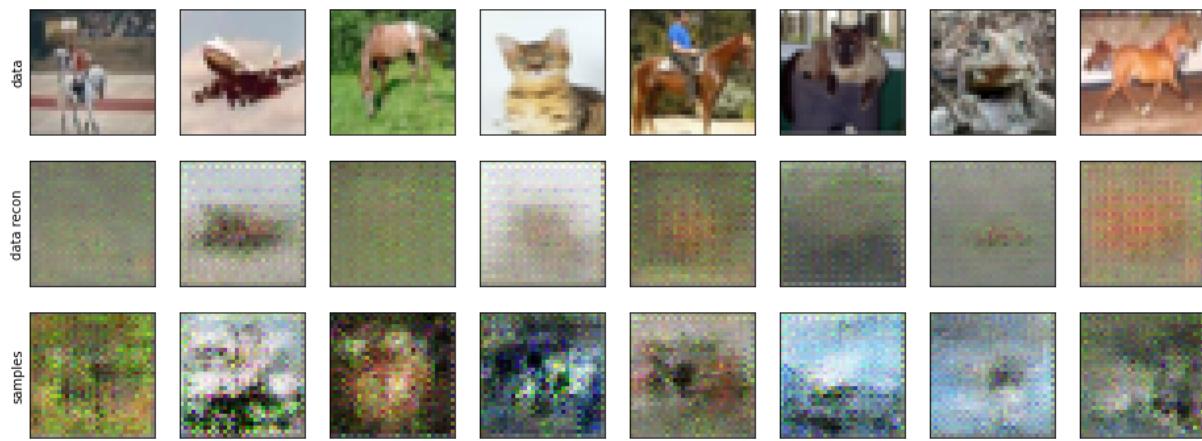


Figura 34. Resultados obtenidos con el modelo VAE-GAN con 10 épocas.

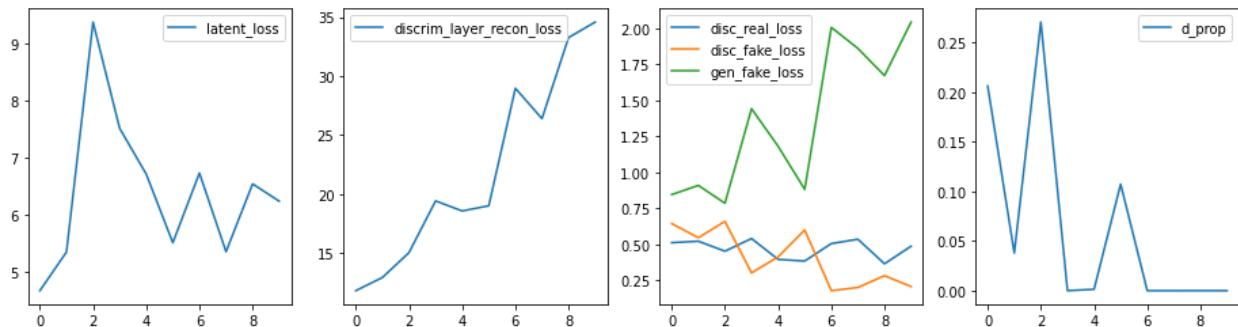


Figura 35. Gráficas obtenidas del Modelo VAE-GAN con 10 épocas.

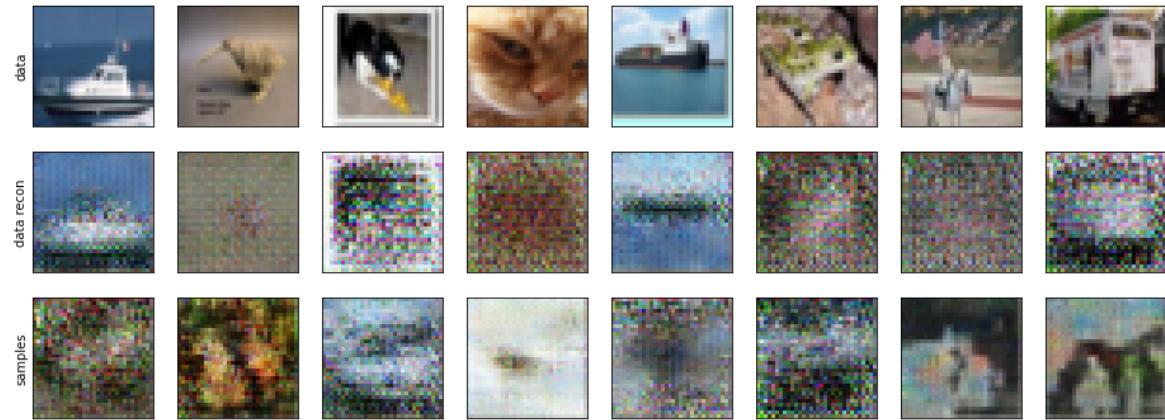


Figura 36. Resultados obtenidos con el modelo VAE-GAN con 50 épocas.

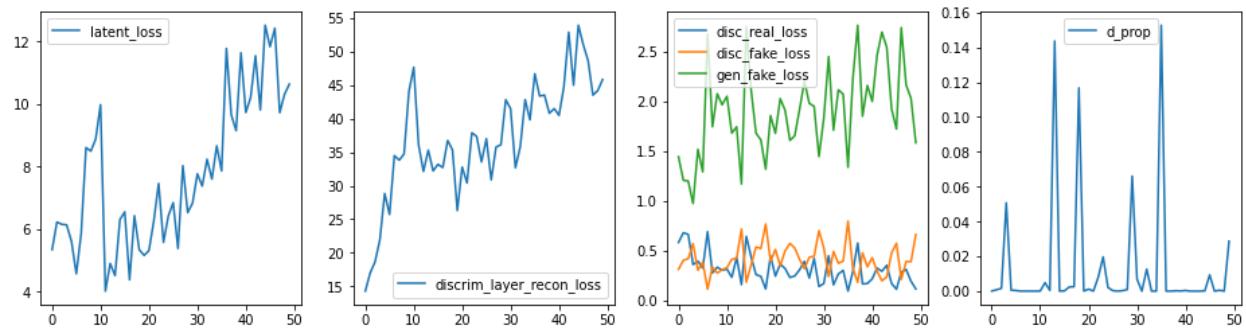


Figura 37. Gráficas obtenidas del Modelo VAE-GAN con 50 épocas.



Figura 38. Resultados obtenidos con el modelo VAE-GAN con 100 épocas.

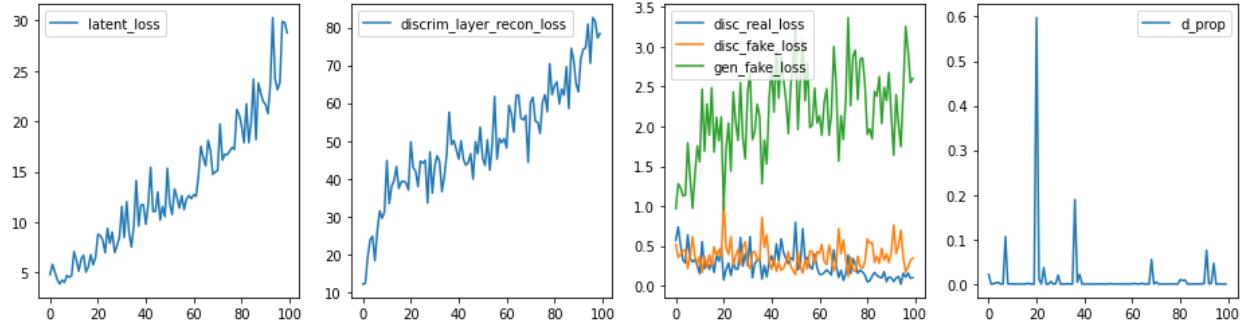


Figura 39. Gráficas obtenidas del Modelo VAE-GAN con 100 épocas.

## Generative adversarial interpolative autoencoder (GAIA)

En el caso de este modelo se tuvieron ciertos inconvenientes al querer proceder con su correspondiente ejecución, por lo cual a continuación se muestran varias figuras, en donde se pueden observar los diferentes errores obtenidos así como sus propuestas de solución.

En la Figura 40 se muestra el primer error obtenido, y al cual se le buscaron distintas soluciones.

```

n_epochs = 50
for epoch in range(n_epochs):
    # train
    for batch, train_x in tqdm(
        zip(range(N_TRAIN_BATCHES), train_dataset), total=N_TRAIN_BATCHES
    ):
        model.train(train_x)
    # test on holdout
    loss = []
    for batch, test_x in tqdm(
        zip(range(N_TEST_BATCHES), test_dataset), total=N_TEST_BATCHES
    ):
        loss.append(model.compute_loss(test_x))
    losses.loc[epoch, 'losses'] = np.mean(loss, axis=0)
# plot results
display.clear_output()
print(
    "Epoch: {} | recon_loss: {} | latent_loss: {}".format(
        epoch, losses.recon_loss.values[-1], losses.latent_loss.values[-1]
    )
)
plot_reconstruction(model, example_data)

```

TypeError: in user code:

```

File "/cipython-input-6-543187205ba0x", line 50, in train
    gradients = self.compute_gradients(train_x)
File "/cipython-input-6-543187205ba0x", line 45, in compute_gradients
    loss = self.compute_loss(x)
File "/cipython-input-6-543187205ba0x", line 32, in compute_loss
    t = q_z.sample()
File "/usr/local/lib/python3.7/dist-packages/torch/distributions/transformed_distribution.py", line 840, in sample
    ..._sample_fn(self._parameters["seed"], seed, name, **kwargs)
File "/usr/local/lib/python3.7/dist-packages/torch/distributions/transformed_distribution.py", line 390, in _call_sample_fn
    self._bijector.forward(x, **bijector_kwarg)
File "/usr/local/lib/python3.7/dist-packages/tensorflow_probability/python/bijectors/bijector.py",
line 933, in forward
    return self._call_forward(x, name, **kwarg)
File "/usr/local/lib/python3.7/dist-packages/tensorflow_probability/python/bijectors/bijector.py",
line 900, in _call_forward
    mapping = self._lookup(x=x, kwargs=kwargs)
File "/usr/local/lib/python3.7/dist-packages/tensorflow_probability/python/bijectors/bijector.py",
line 1343, in _lookup
    mapping = self._from_x(x).get(subkey, mapping).merge(x=x)
File "/usr/local/lib/python3.7/dist-packages/tensorflow_probability/python/bijectors/bijector.py",
line 151, in __getitem__
    return super().__getitem__(key)
File "/usr/local/lib/python3.7/dist-packages/tensorflow_probability/python/bijectors/bijector.py",
line 181, in __hash__
    return hash(x)

```

TypeError: Tensor is unhashable. Instead, use tensor.ref() as the key.

SEARCH STACK OVERFLOW

Figura 40. Error obtenido en la etapa de entrenamiento del Modelo GAIA.

Se intentó añadiendo las siguiente líneas de código, también utilizadas en los demás modelos.

```
import tensorflow.compat.v1.keras.backend as K
import tensorflow as tf
tf.compat.v1.disable_eager_execution()

tf.compat.v1.disable_tensor_equality()
```

Figura 41. Líneas añadidas para la ejecución del modelo propuesto.

Sin embargo, en la Figura 42 se puede visualizar que aparece un error diferente después de implementar las líneas anteriores de la Figura 41.

```
StagingError: in user code:

  File "<ipython-input-6-dcb8c3de5f5c>", line 85, in train  *
      gen_gradients, disc_gradients = self.compute_gradients(x)
  File "<ipython-input-6-dcb8c3de5f5c>", line 56, in compute_gradients  *
      d_xg_loss, d_xi_loss, d_xi_loss, xg_loss = self.compute_loss(x)

    LookupError: No gradient defined for
operation'gradients/model/up_sampling2d/resize_1/ResizeNearestNeighbor_grad/ResizeNearestNeighborGrad' (op
type: ResizeNearestNeighborGrad). In general every operation must have an associated `@tf.RegisterGradient` for
correct autodiff, which this op is lacking. If you want to pretend this operation is a constant in your
program, you may insert `tf.stop_gradient`. This can be useful to silence the error in cases where you know
gradients are not needed, e.g. the forward pass of tf.custom_gradient. Please see more details in
https://www.tensorflow.org/api\_docs/python/tf/custom\_gradient.
```

SEARCH STACK OVERFLOW

Figura 42. Errores obtenidos después de añadir las líneas para solucionar la incompatibilidad de las librerías de Tensorflow.

Nuevamente, buscando documentación sobre este tipo de errores se encontró una posible solución: utilizar la función de tensorflow `tf.stop_gradient()`. Sin embargo, de nueva cuenta no funcionó.

```
@tf.function
def compute_gradients(self, x):
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        d_xg_loss, d_xi_loss, d_x_loss, xg_loss = self.compute_loss(x)
        d_xg_loss = tf.stop_gradient(d_xg_loss)
        d_xi_loss = tf.stop_gradient(d_xi_loss)
        d_x_loss = tf.stop_gradient(d_x_loss)
        xg_loss = tf.stop_gradient(xg_loss)

        gen_loss = d_xg_loss + d_xi_loss
        disc_loss = d_xg_loss + d_x_loss - tf.clip_by_value(d_xi_loss, 0, d_x_loss)

    gen_gradients = gen_tape.gradient(
        gen_loss, self.enc.trainable_variables + self.dec.trainable_variables
    )
    disc_gradients = disc_tape.gradient(disc_loss, self.disc.trainable_variables)
    return gen_gradients, disc_gradients
```

Figura 43. Líneas añadidas para solucionar los errores anteriores.

```
@tf.function
def train(self, x):
    gen_gradients, disc_gradients = self.compute_gradients(x)
    self.apply_gradients(gen_gradients, disc_gradients)
    gen_gradients = tf.stop_gradient(gen_gradients)
    disc_gradients = tf.stop_gradient(disc_gradients)
```

Figura 45. Líneas añadidas para solucionar los errores anteriores.

## Análisis de los resultados

En esta sección se realizará una comparación de los resultados obtenidos al realizar la implementación de cada uno de los modelos propuestos, por lo cual a continuación se podrán visualizar las siguientes tablas comparativas.

TABLA 8  
Comparación de resultados obtenidos de los modelos propuestos para la época 50

| Modelos generativos                                                   | Gen_optimizer | Disc_optimizer | Pérdida discriminador                                                           | Pérdida generador  |
|-----------------------------------------------------------------------|---------------|----------------|---------------------------------------------------------------------------------|--------------------|
| Generative Adversarial Network(GAN)                                   | 0.001         | 0.005          | 1.3866125345230103                                                              | 0.7030718922615051 |
|                                                                       | 0.01          | 0.05           | 9.96210780562128e-15                                                            | 32.2401237487793   |
|                                                                       | 0.0001        | 0.0005         | 9.96580341037015e-15                                                            | 32.239994049072266 |
| Variational Autoencoder with Generative Adversarial Network (VAE-GAN) | 0.0001        | 0.0001         | disc_fake_loss:<br>0.6593310236930847<br>disc_real_loss:<br>0.11215417832136154 | 1.581476092338562  |

TABLA 9  
Comparación de resultados obtenidos de los modelos propuestos para la época 100

| Modelos generativos                             | Gen_optimizer | Disc_optimizer | Pérdida discriminador | Pérdida generador  |
|-------------------------------------------------|---------------|----------------|-----------------------|--------------------|
| Generative Adversarial Network(GAN)             | 0.001         | 0.005          | 1.3862947225570679    | 0.6931473612785339 |
|                                                 | 0.01          | 0.05           | 1.3864511251449585    | 0.7057249546051025 |
|                                                 | 0.0001        | 0.0005         | 1.1299501657485962    | 0.8322003483772278 |
| Wasserstein GAN with Gradient Penalty (WGAN-GP) | 0.0001        | 0.0005         | nan                   | nan                |

|                                                                       |        |        |                                                                                 |                   |
|-----------------------------------------------------------------------|--------|--------|---------------------------------------------------------------------------------|-------------------|
| Variational Autoencoder with Generative Adversarial Network (VAE-GAN) | 0.0001 | 0.0001 | disc_fake_loss:<br>0.3464422821998596<br>disc_real_loss:<br>0.10107631236314774 | 2.602600574493408 |
|-----------------------------------------------------------------------|--------|--------|---------------------------------------------------------------------------------|-------------------|

TABLA 10  
Comparativa de los tiempos de ejecución de los modelos propuestos

| Modelos                                                               | Épocas | Tiempo       |
|-----------------------------------------------------------------------|--------|--------------|
| Autoenconder(AE)                                                      | 10     | 50 seg       |
|                                                                       | 50     | 3 min        |
|                                                                       | 150    | 13 min       |
| Variational Autoenconder (VAE)                                        | 10     | 5 min        |
|                                                                       | 50     |              |
|                                                                       | 100    |              |
| Generative Adversarial Network(GAN)                                   | 50     | 4 min        |
|                                                                       | 100    | 8 min        |
| Wasserstein GAN with Gradient Penalty (WGAN-GP)                       | 10     | 20 min       |
|                                                                       | 20     | 1 hr         |
|                                                                       | 100    | 3 hrs.       |
| Variational Autoencoder with Generative Adversarial Network (VAE-GAN) | 10     | 10min        |
|                                                                       | 50     | 1 hr.        |
|                                                                       | 100    | 1 hr 30 min. |
| Generative adversarial interpolative autoencoder(GAIA)                | --     | --           |

## Conclusiones

La realización de esta tarea se enfocó en conocer el comportamiento de los modelos propuestos basados en autocodificadores, modelos generativos, así como su correspondientes combinaciones; en donde para ello fue necesario la selección de una base de datos, que nos permitiera observar los posibles resultados al variar ciertos parámetros como son el número de épocas, el valor de los optimizadores, el tamaño del batch y más.

Así bien, como se pudo apreciar en la TABLA 1 fue necesario la modificación de ciertos parámetros de entrada de cada uno de los modelos, como fue ajuste en las dimensiones de los datos de entrada debido a que se cuenta con imágenes a color de tamaño 32x32x3; con el fin de poder realizar su correspondiente ejecución.

Por otra parte, en la sección de resultados fue posible observar que la mayoría de los modelos implementados dieron como resultados imágenes un tanto borrosas aunque era posible identificar que tenían ciertas similitudes con las imágenes de entrada debido a que se podía apreciar las siluetas de los objetos, así bien se conservó su correspondiente paleta de colores, como fue el caso del modelo AE y el modelo VAE-GAN.

Por el contrario, para el caso de los modelos VAE, GAN,y WGAN-GP, en donde para ciertos valores de los optimizadores sus imágenes resultantes no mostraron tener similitudes con las imágenes de entrada debido a que mostraron imágenes pixeleadadas en una paleta de colores distinta a la paleta original, todo esto a pesar de que se probó con distintas épocas de entrenamiento.

## Referencias

- [1] Autoencoder Recuperado el día 05 de Junio de 2021 de <https://www.sciencedirect.com/topics/engineering/autoencoder>
- [2] Mishra, S. (2021) An Introduction to VAE-GANs. Recuperado el día 5 de Junio de 2022 de <https://wandb.ai/shambhavicodes/vae-gan/reports/An-Introduction-to-VAE-GANs--VmIldzoxMTcxMjM5>
- [3] Mi, L., Shen, M., Zhang, J. (2018) A Probe Towards Understanding GAN and VAE Models. Recuperado el día 5 de Junio de 2022 de <https://arxiv.org/pdf/1812.05676.pdf>
- [3] ReposHub.Implementations of a number of generative models in Tensorflow 2. Gan, VAE.Seq2Seq, VAEGAN, GAIA, Spectrogram Inversion. Everything is self contained in a jupyter notebook for easy export to colab. Recuperado el día 01 de Junio de 2022 de <https://rephub.com/python/deep-learning/timsainb-tensorflow2-generative-models.html>
- [4] Sainburg, T.(2018) Generative Adversarial Interpolative Autoencoding(GAIA). Recupeado el 05 de Junio de <https://timsainburg.com/gaia.html>