UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA
DIVISIÓN DE INVESTIGACIÓN Y POSGRADO

GAN: aplicación en la generación de imágenes de Medusas

Autor:
Act. Edgar Adrian Quiroz Calvillo

8 de marzo de 2024

Índice

I. Resumen

Las Redes Generativas Adversariales (GANs) han emergido como una poderosa herramienta en el campo del aprendizaje profundo para la generación de datos realistas. En este trabajo, exploramos el potencial de las GANs desde una perspectiva educativa y experimental, comenzando con la generación de puntos en una línea recta y avanzando hacia la aproximación de una función coseno. Utilizando una arquitectura de red GAN, demostramos cómo estas redes pueden aprender la distribución de datos subyacente y generar muestras que se asemejen a la distribución de datos reales. A través de iteraciones de entrenamiento y ajustes de parámetros, observamos cómo la red GAN evoluciona desde la generación de patrones simples hacia la captura de patrones más complejos. Este estudio busca generar imágenes nuevas basado en un conjunto de datos que comprende $1,021$ imágenes de medusas pertenecientes a seis categorías y especies distintas: medusa mauve stinger, medusa moon, medusa barrel, medusa blue, medusa compass, y medusa lion's mane.

II. Antecedentes

Las Redes Generativas Adversariales (GANs) son un tipo de modelo generativo introducido por [1], que consta de dos redes neuronales principales: el generador y el discriminador. El generador toma una distribución de ruido aleatorio como entrada y genera muestras de datos que se asemejan a las observaciones reales. Por otro lado, el discriminador intenta distinguir entre las muestras generadas por el generador y las muestras reales del conjunto de datos. Ambas redes se entrenan de manera adversarial, es decir, en un juego de suma cero, donde el generador busca engañar al discriminador y viceversa.

a. Principios de Aprendizaje en GANs

El entrenamiento de una GAN implica encontrar un equilibrio entre el generador y el discriminador, donde ninguno de los dos domine completamente al otro. [2]Esto se logra a través de la optimización de dos funciones de pérdida: la pérdida del generador, que busca minimizar la probabilidad de que el discriminador identifique incorrectamente las muestras generadas como falsificaciones, y la pérdida del discriminador, que busca maximizar la probabilidad de clasificar correctamente entre muestras reales y falsificadas. [3]Este proceso iterativo de ajuste de parámetros permite que la GAN aprenda a generar muestras que sean indistinguibles de las muestras reales.

En resumen una red GAN aprende cómo capturar la distribución subyacente de los datos de entrenamiento y generar nuevos datos que sean consistentes con esta distribución.

b. Descripción de las Especies de Medusas

a. Medusa Moon (Aurelia aurita)

La medusa moon es una especie común con cuatro gónadas en forma de herradura, visibles a través de la parte superior de su campana translúcida. Su método de alimentación involucra la recolección de medusas, plancton y moluscos mediante sus tentáculos.

Figura 1: Medusa Moon

b. Medusa Barrel (Rhizostoma pulmo)

La medusa barrel ostenta el título de ser la medusa más grande encontrada en aguas británicas, con una campana que puede alcanzar hasta 90 cm de diámetro. Se alimenta de plancton y pequeños peces, atrapándolos con sus tentáculos.

Figura 2: Medusa barrel

c. Medusa Blue (Cyanea lamarckii)

La medusa blue es de gran tamaño, con la capacidad de crecer hasta 30 cm de diámetro. Su método de alimentación implica capturar plancton y pequeños peces con sus tentáculos.

Figura 3: Medusa blue

d. Medusa Compass (Chrysaora hysoscella)

Denominada así por las marcas marrones en su campana que se asemejan a una rosa de los vientos, la medusa compass se alimenta de plancton y pequeños peces atrapándolos con sus tentáculos.

Figura 4: Medusa compass

e. Medusa Lion's Mane (Cyanea capillata)

La medusa lion's mane ostenta el título de ser la medusa más grande del mundo, con una campana que puede alcanzar hasta 2 metros de diámetro y tentáculos que se extienden hasta

30 metros de longitud. Su método de alimentación implica capturar plancton y pequeños peces con sus tentáculos.

Figura 5: Medusa Lion's Mane

f. Medusa Mauve Stinger (Pelagia noctiluca)

La medusa mauve stinger, de pequeño tamaño, presenta tentáculos largos y estructuras verrugosas en su campana llenas de células urticantes. Se alimenta de otras medusas pequeñas y ascidias marinas oceánicas.

Figura 6: Medusa Mauve Stinger

III. Caso de estudio

El caso de uso propuesto implica la utilización de una red GAN para la generación imágenes de medusas basado en diferentes características de 6 especies de medusas.

IV. Generador

El Generador de imágenes tendrá como entrada un espacio latente de 100 y contará con la siguiente arquitectura:

Tipo de capa	Características	Función de activación
Dense	Output units = 4*4*256	ReLu
Reshape	$(4, 4, 256)$	
UpSampling2D	$Factor = 2$	
Conv2D	$filtros = 256,\ kernel = 3X3,\ Padding = Same$	ReLu
BatchNormalization	$Momentum = 0{,}8$	
Dense	Output units = 4*4*256	ReLu
Reshape	$(4, 4, 256)$	
UpSampling2D	$Factor = 2$	
Conv2D	$filtros = 256,\ kernel = 3X3,\ Padding = Same$	ReLu
BatchNormalization	$Momentum = 0{,}8$	
Dense	Output units = 4*4*256	ReLu
Reshape	$(4, 4, 256)$	
UpSampling2D	$Factor = 2$	
Conv2D	$filtros = 256,\ kernel = 3X3,\ Padding = Same$	ReLu
BatchNormalization	$Momentum = 0{,}8$	
Dense	Output units = 4*4*256	ReLu
Reshape	$(4, 4, 256)$	
UpSampling2D	$Factor = 2$	
Conv2D	$filtros = 256,\ kernel = 3X3,\ Padding = Same$	ReLu
BatchNormalization	$Momentum = 0{,}8$	
Conv2D	$filtros = 3,\ kernel = 3X3,\ Padding = Same$	Tanh

Cuadro 1: Arquitectura del generador.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 4096)              413696

 reshape (Reshape)           (None, 4, 4, 256)         0

 up_sampling2d (UpSampling2  (None, 8, 8, 256)         0
 D)

 conv2d (Conv2D)             (None, 8, 8, 256)         590080

 batch_normalization (Batch  (None, 8, 8, 256)         1024
 Normalization)

 activation (Activation)     (None, 8, 8, 256)         0

 up_sampling2d_1 (UpSamplin  (None, 16, 16, 256)       0
 g2D)

 conv2d_1 (Conv2D)           (None, 16, 16, 256)       590080

 batch_normalization_1 (Bat  (None, 16, 16, 256)       1024
 chNormalization)

 activation_1 (Activation)   (None, 16, 16, 256)       0

 up_sampling2d_2 (UpSamplin  (None, 32, 32, 256)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 32, 32, 256)       590080

 batch_normalization_2 (Bat  (None, 32, 32, 256)       1024
 chNormalization)

 activation_2 (Activation)   (None, 32, 32, 256)       0

 up_sampling2d_3 (UpSamplin  (None, 64, 64, 256)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 64, 64, 128)       295040

 batch_normalization_3 (Bat  (None, 64, 64, 128)       512
 chNormalization)

 activation_3 (Activation)   (None, 64, 64, 128)       0

 conv2d_4 (Conv2D)           (None, 64, 64, 3)         3459

 activation_4 (Activation)   (None, 64, 64, 3)         0

=================================================================
Total params: 2486019 (9.48 MB)
Trainable params: 2484227 (9.48 MB)
Non-trainable params: 1792 (7.00 KB)
_________________________________________________________________
```

Figura 7: Arquitectura del Generador

V. Discriminador

Se define la arquitectura del discriminador como:

Tipo de capa	Características	Función de ac
Conv2D	$filtros = 32$, $kernel = 3X3$, $Padding = Same$, $input = (64, 64, 3)$	LeakyReLU a
Dropout	0.25	
Conv2D	$filtros = 64$, $kernel = 3X3$, $Padding = Same$	LeakyReLU a
ZeroPadding2D	padding=((0,1),(0,1))	
BatchNormalization	$Momentum = 0{,}8$	
Dropout	0.25	
Conv2D	$filtros = 128$, $kernel = 3X3$, $Padding = Same$	LeakyReLU a
Conv2D	$filtros = 256$, $kernel = 3X3$, $Padding = Same$	LeakyReLU a
Conv2D	$filtros = 512$, $kernel = 3X3$, $Padding = Same$	LeakyReLU a
Flatten		
Dense	1	Sigmoi

Cuadro 2: Arquitectura del discriminador.

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 32, 32, 32)        896

 leaky_re_lu_5 (LeakyReLU)   (None, 32, 32, 32)        0

 dropout_5 (Dropout)         (None, 32, 32, 32)        0

 conv2d_11 (Conv2D)          (None, 16, 16, 64)        18496

 zero_padding2d_1 (ZeroPadd  (None, 17, 17, 64)        0
 ing2D)

 batch_normalization_8 (Bat  (None, 17, 17, 64)        256
 chNormalization)

 leaky_re_lu_6 (LeakyReLU)   (None, 17, 17, 64)        0

 dropout_6 (Dropout)         (None, 17, 17, 64)        0

 conv2d_12 (Conv2D)          (None, 9, 9, 128)         73856

 batch_normalization_9 (Bat  (None, 9, 9, 128)         512
 chNormalization)

 leaky_re_lu_7 (LeakyReLU)   (None, 9, 9, 128)         0

 dropout_7 (Dropout)         (None, 9, 9, 128)         0

 conv2d_13 (Conv2D)          (None, 9, 9, 256)         295168

 batch_normalization_10 (Ba  (None, 9, 9, 256)         1024
 tchNormalization)

 leaky_re_lu_8 (LeakyReLU)   (None, 9, 9, 256)         0

 dropout_8 (Dropout)         (None, 9, 9, 256)         0

 conv2d_14 (Conv2D)          (None, 9, 9, 512)         1180160

 batch_normalization_11 (Ba  (None, 9, 9, 512)         2048
 tchNormalization)

 leaky_re_lu_9 (LeakyReLU)   (None, 9, 9, 512)         0

 dropout_9 (Dropout)         (None, 9, 9, 512)         0

 flatten_1 (Flatten)         (None, 41472)             0

 dense_2 (Dense)             (None, 1)                 41473

=================================================================
Total params: 1613889 (6.16 MB)
Trainable params: 1611969 (6.15 MB)
Non-trainable params: 1920 (7.50 KB)
_________________________________________________________________
```

Figura 8: Arquitectura del Discriminador

VI. GAN

Se genera la red GAN como una clase de python con los siguientes métodos:

a. __init__

Este método inicializa la clase GAN y recibe tres argumentos: discriminator, generator y latent_dim. Estos son el discriminador, el generador y la dimensión latente respectivamente.

b. compile

Este método configura la GAN para el entrenamiento. Recibe tres argumentos: d_optimizer, g_optimizer y loss_fn, que son los optimizadores para el discriminador y el generador, y la función de pérdida respectivamente. Además, inicializa las métricas de pérdida del discriminador y el generador.

c. metrics

Este método es un decorador @property que devuelve una lista de métricas, que en este caso son las pérdidas del discriminador y el generador.

d. train_step

Este método define un paso de entrenamiento personalizado para la GAN. Recibe real_images como entrada, que son las imágenes reales del conjunto de datos de entrenamiento. Dentro del método, se realizan las siguientes operaciones:

- Se generan imágenes falsas a partir de vectores de ruido en el espacio latente utilizando el generador.
- Se combinan las imágenes falsas generadas y las imágenes reales.
- Se crean etiquetas para diferenciar entre imágenes reales y falsas.
- Se entrena el discriminador para distinguir entre imágenes reales y falsas.
- Se genera ruido en el espacio latente y se crean etiquetas engañosas para entrenar el generador.
- Se entrena el generador para engañar al discriminador.
- Se actualizan las métricas de pérdida del discriminador y el generador.
- Se devuelve un diccionario con las pérdidas del discriminador y el generador.

VII. Resultados

Después de un entrenamiento de 100 epochs se obtuvo un resultado satisfactorio con la red GAN. en la gráfica learning curve se puede ver como iterara la pérdida tanto del discriminador como del generador. Esta oscilación representa como van aprendiendo cada una de las redes representando una pérdida para la otra.

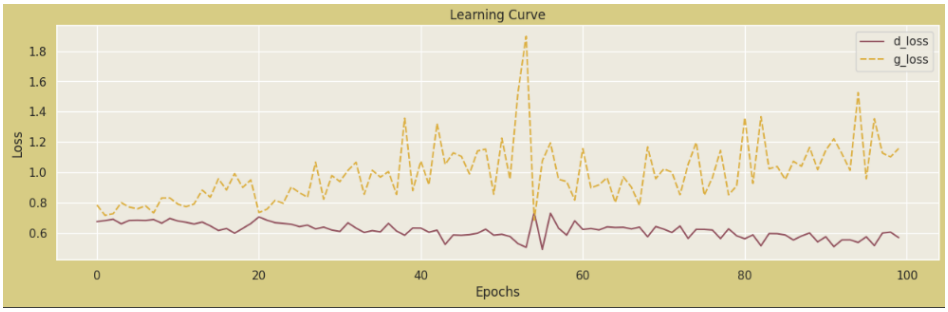

Figura 9: Learning curve

Algunas imágenes generadas por la red son las siguientes:

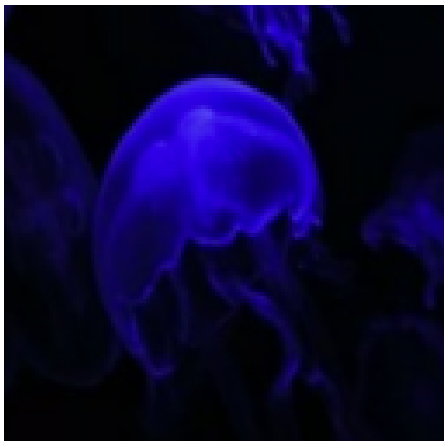

Figura 10: Imagen generada 1

Pueden encontrar el código en el siguiente repositorio: `https://github.com/Eddqzc/GAN_jellyfish_images`

Figura 11: Imagen generada 2

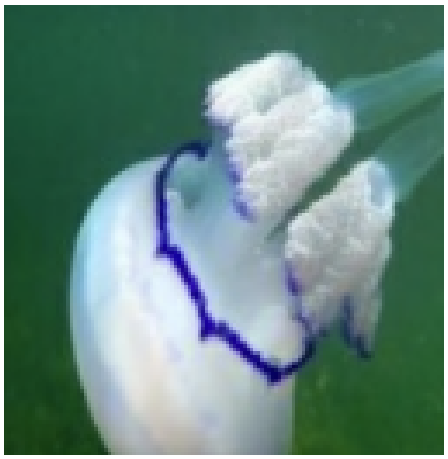

Figura 12: Imagen generada 3

Referencias bibliográficas

[1] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[2] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, "A review on generative adversarial networks: Algorithms, theory, and applications," *IEEE transactions on knowledge and data engineering*, vol. 35, no. 4, pp. 3313–3332, 2021.

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*,

vol. 63, no. 11, pp. 139–144, 2020.