Proyecto: Eliminación de Ruido con Muestreo de Gibbs

31508811-1 Martiñón Luna Jonathan José.

***Abstract*—Basado en towardsdatascience.com. En diversas ocasiones nos hemos encontrado con imágenes que por algún motivo no resultan completamente nítidas, existe uno que otro ruido que no permite apreciarla en su mejor calidad.**

A través del presente proyecto, se pone en práctica el muestreo de gibbs, con el objetivo de poder llevar a cabo la eliminación de ruido para una imágen en blanco y negro con 2 valores internos (0 o 1).

I. Objetivos

- Realizar una *depuración* de ruido presente en imágenes. De tal forma que se consiga un mejor entendimiento del contenido de la misma.
- Comparar el rendimiento del algoritmo al tratarse de distintos tipos de imágenes (Con fondo blanco, sin fondo, con fondo cuadriculado).
- Comparar el rendimiento ante la presencia de ruido al 25% y 5%

Resulta importante mencionar, que los valores internos se ajustaron a -1 y 1, donde -1 correspondería a valores menores o iguales a 0.5, mientras que 1 correspondería a valores mayores a 0.5. De igual forma, se agregó un padding de 1, para evitar problemas de esquinas

II. Ambiente

La presente práctica se desarrollará en *Colab*, haciendo uso de Python 3.7.10; en conjunto con las librerías *numpy* y *matplotlib*. La dirección del repositorio de Gihub con el código y los datos es:

https://github.com/JonathanMartignon/LicenciaturaCienciaDeDatos/tree/main/Computacion_Estadistica/Proyecto

III. Introducción

EL presente proyecto buscará implementar el muestreo de gibbs con el objetivo de eliminar el *ruido* presente en imágenes. Para ello se utilizarán 3 imágenes específicas:

Fig. 1. Mariposa.
La presente imagen fue recuperada de: https://www.pinterest.com/pin/500251471110762031/.
El objetivo de seleccionar la presente imagen es observar el comportamiento del algoritmo cuando existe una mayoría de fondo blanco.

Fig. 2. Pericos.
La presente imagen fue recuperada de: https://www.mascotahogar.com/imagenes-wallpaper-artistico-periquitos-blanco-negro-jpg-1024x768.
El objetivo de seleccionar la presente imagen es observar el comportamiento del algoritmo cuando realmente no existe un fondo blanco como tal.

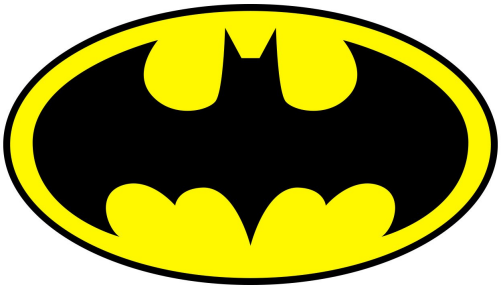

Fig. 3. Logo de Batman.
La presente imagen fue recuperada de: https://1000marcas.net/logo-batman/.
El objetivo de seleccionar la presente imagen es aprovechar la presencia de únicamente 2 tipos de valores (en apariencia) 0 y 1.

Tal como se puede apreciar en las figuras anteriores (1, 2 y 3), las imágenes no tienen presencia de ruido el cuál eliminar, es por ello que serán sometidas a un pre-procesamiento dónde se les añadirá ruido blanco, que posteriormente procederemos

a tratar de eliminar.

Finalmente, buscaremos comparar el rendimiento en la eliminación de ruido resultante para cada imagen, donde esperamos, se vea afectada lo menor posible y *restaurada* en la mejor forma.

IV. Métodologías

A. MCMC (Marcov Chain Monte Carlo)

Existen ocasiones en que no es posible (O resulta muy complicado) realizar el cálculo exacto de una integral, es por ello que se recurre a métodos de muestreo para aproximarla. Uno de esos métodos es el método de Monte Carlo; sin embargo, resulta complicado para distribuciones complejas, es por ello que surge el método de *Monte Carlo basado en Cadenas de Markov*. Uno de los algoritmos más utilizados en estos casos es *Metrópolis-Hastings* [1]

B. Metrópolis-Hastings

Se requiere poder calcular el producto de la inicial y la verosimilitud, hasta obtener una constante de proporcionalidad. El método produce una aproximación de la distribución posterior $p(\theta|x)$, mediante una muestra de valores θ obtenidos de dicha distribución. [2]

Podemos simplificar los pasos de la siguiente forma: [3]

1) Inicializamos $X_1 = x_1$
2) Para $t = 1, 2, ...$ Generamos un valor y a partir de $Q(y|x_t)$ donde Q es el kernel de transición y y es el candidato para el valor x_{t+1}
3) Calcular la probabilidad de aceptación (A):

$$A = min\left(1, \frac{\pi(y)Q(x_t|y)}{\pi(x_t)Q(y|x_t)}\right)$$

4) Si se acepta la probabilidad A, entonces establecemos $x_{t+1} = y$. En otro caso mantenemos $x_{t+1} = x$

C. Muestreo de Gibbs

A grandes rasgos, podremos decir que resulta en un caso especial del algoritmo Metrópolis-Hastings, por lo tanto, también del MCMC. Es un algoritmo para generar una muestra aleatoria a partir de la distribución de probabilidad conjunta de dos o más variables aleatorias. [4]

Se trata de una *caminata aleatoria*, para la cual se inicia en un punto aleatorio, mientras que el movimiento próximo dependerá únicamente de la posición actual. Permite generar muestras de la posterior $p(\theta_1, ..., \theta_p|x)$ Siempre y cuando se puedan generar valores de todas las probabilidades condicionales $p(\theta_k|\theta_1, ..., \theta_{k-1}, \theta_{k+1}, ..., \theta_p, x)$. [5]

Podemos definir el proceso de la siguiente forma:

1) Iniciamos en el parámetro θ_k, generamos una simulación de distribución condicional de la forma: $p(\theta_k|\theta_1, ..., \theta_{k-1}, \theta_{k+1}, ..., \theta_p, x)$
2) La nueva posición estará dada por el nuevo valor θ_k y los valores que no han cambiado $\theta_1, ..., \theta_{k-1}, \theta_{k+1}, ..., \theta_p$
3) Se selecciona un nuevo componente θ_{k+1} y se repite el paso 1.

V. Pre-procesamiento

Para poder comparar las imágenes y el rendimiento, primero debemos asegurarnos que se encuentren en condiciones lo más parecidas posibles, por lo que iniciaremos hablando del tamaño.

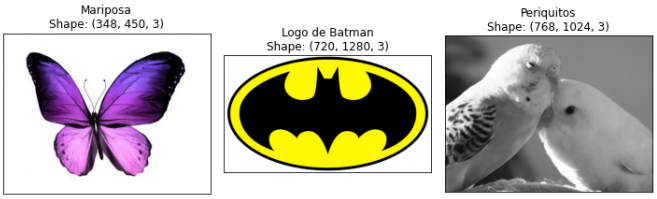

Fig. 4. Diferencia de Tamaños.
En la presente imagen se puede apreciar el comparativo entre las 3 imágenes, mostrando características en el orden (Alto,Ancho,Canales).

Tal cual podemos apreciar en la figura 4, cada una de las imagenes presenta un tamaño distinto. Si somos más meticulosos, podremos darnos cuenta que la imagen de menor tamaño resulta en la mariposa, motivo por el cual, ajustaremos las imágenes a un tamaño similar (340,450).

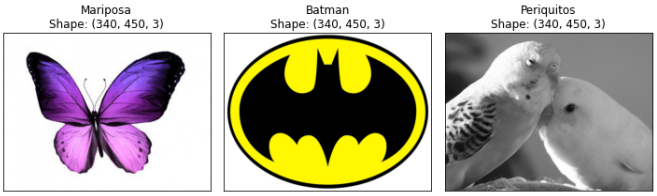

Fig. 5. Ajuste de formas.
Ajuste de forma en las imágenes; mostrando características en el orden (Alto,Ancho,Canales).

Tal cual podemos apreciar en la figura 5, las imágenes han sido ajustadas para tener el mismo valor en alto y ancho (340 y 450 respectivamente). Si nos damos cuenta, existe un valor que aún nos está generando *ruido*, es el total de canales. Existen 3 canales, debido a que se trata de una imagen RGB. por lo que procederemos también a reducirlos a 1 sólo canal.

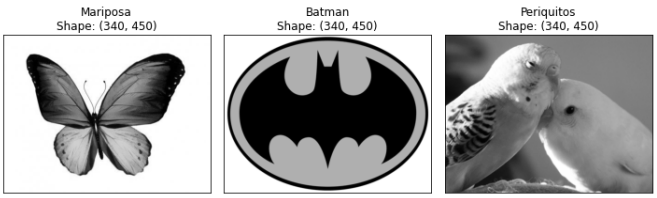

Fig. 6. Reducción de canales.
Reducción de 3 a 1 sólo canal, teniendo como consecuente una imagen en blanco y negro, mostrando características en el orden (Alto,Ancho,Canales).

Hemos estandarizado las imágenes en forma (ancho, alto y canales), falta el motivo de estar aquí, hay que generar el ruido que hemos de eliminar. Para la incrustación de ruido, se seleccionó un porcentaje de la imagen a aplicar, posterior a eso, a pixeles aleatorios se les generó un número aleatorio entre 0 o 1, para simular el ruido. Inicialmente se optó por generarlos con un 50% de ruido, sin embargo, la imagen parecía que estaba saturada del mismo, tal cual se puede apreciar en la figura 7.

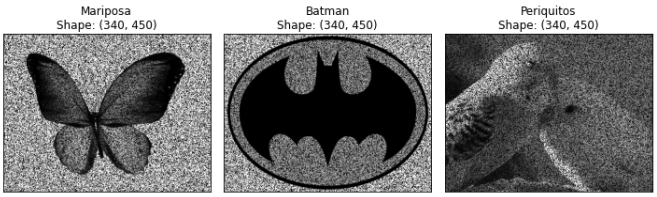

Fig. 7. Generación de ruido.
Generación de ruido tratando de abarcar un 50% de la imagen original.

Teniendo en cuenta lo anterior, se procedió a generar ruido en un total de 25% de la imagen, lo que resultó bastante más accesible, (Figura 8).

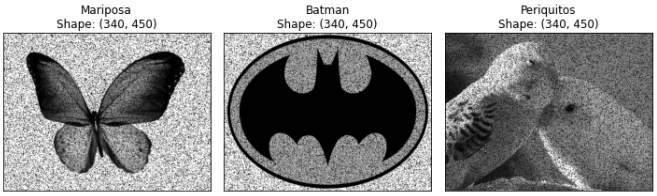

Fig. 8. Generación de ruido.
Generación de ruido tratando de abarcar un 25% de la imagen original.

Sin embargo, dada la diferencia de ruido observada en los intentos con 25% y 50%, se decidió agregar otro conjunto, en el cual las imágenes presentaron un 5% de ruido. De esa forma el ruido máximo no saturó la imagen y el ruido mínimo permitía comparar rendimientos. Finalmente, los 2 conjuntos a trabajar se presentan en la figura 9.

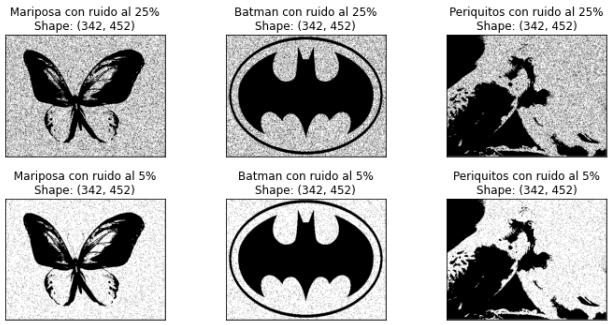

Fig. 9. Conjuntos Finales.
Comparación entre los 2 conjuntos finales, quienes presentan un ruido del 25% y 5%.

VI. Especificando el modelo [6]

Sabemos que poseemos una imagen con ruido X, que a su vez, puede ser vista como una matriz $X = matriz_{nxm}$, donde cada pixel $x_{i,j} \in \{-1, 1\}$ puesto que así hemos definido los valores.

Por otro lado, existe otra matriz (imagen) Y que es nuestra matriz objetivo, libre de ruido con mismas dimensiones que la matriz X.

Podríamos ver la eliminación de ruido como la probabilidad de calcular la matriz Y a partir de X, es decir: $P(Y|X)$, que por teorema de bayes podemos reescribir como:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$

Claro, que si buscamos eliminar la división, podremos aplicar el logaritmo (y sus respectivas propiedades) para obtener

$$log\left(P(Y|X)\right) = log(P(X|Y)) + log(P(Y)) - log(P(X))$$

Teniendo en cuenta que buscamos maximizar $log(P(Y|X))$, podremos verlo también como minimizar $-log(P(X|Y)) - log(P(Y))$

Nos fijaremos en los 4 próximos vecinos (arriba, abajo, derecha e izquierda) para tomar una decisión de pertenencia (o clase) del pixel. En el caso de los bordes, resultaría difícil tomar alguno de dichos 4 pixeles, es por ello que se procede a agregar un padding, y de esa forma evitar problemas con las esquinas.

Al tener sólo 2 clases (1 y -1), y asumiendo que el valor de y a predecir es 1, tendremos algo de la forma $P(Y = 1|Y_vecinos)$, donde la probabilidad conjunta X y Y es:

$$P(Y, X) = \frac{1}{Z} exp\left(\eta \sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}y_{ij} + \beta \sum_{i'j' \in N(ij)} y_{ij}y_{i'j'}\right)$$

Donde η y β son hiperparámetros y Z es la constante de normalización. $N(ij)$ los vecinos de y_{ij} sin contar a x_{ij}

Ahora, bien, para calcular la posterior, tenemos:

$$P(y_{i,j} = 1|y_{N(ij),x_{ij}}) = \frac{P(y_{ij} = 1, y_{N(ij)}, x_{ij})}{\sum_{y_{ij} \in (0,1)} P(y_{i,j}, y_{N(ij)}, x_{ij})}$$

$$P(y_{i,j} = 1|y_{N(ij),x_{ij}}) = \frac{1}{1 + exp(-2w_{ij})}$$

donde

$$w_{ij} = \eta x_{ij} + \beta \sum_{N(ij)} y_{N(ij)}$$

Y Finalmente podremos ver la pérdida como:

$$-log(P(X|Y)) - log(P(Y)) =$$

$$-\eta \sum_{i=1}^{n} \sum_{j=1}^{m} x_{ij} y_{ij} - \beta \sum_{i'j' \in N(ij)} y_{ij} y_{i'j'}$$

VII. Resultados

Nuestro método tardó un aproximado de 7 minutos y medio para eliminar el ruido presente en cada una de las imágenes. A continuación, podremos observar el comportamiento de la pérdida para cada conjunto durante el período de quemado.

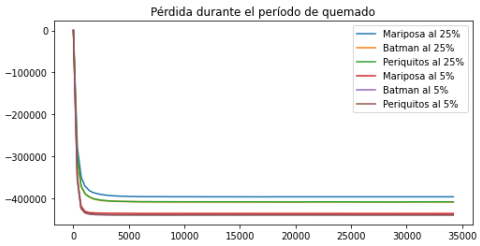

Fig. 10. Período de quemado.
Comportamiento de la pérdida dentro del período de quemado para ambos conjuntos de datos.

Resulta interesante también, comparar una vez que se ha finalizado el período de quemado. Tal cual lo podemos apreciar en la figura 11, existe diferencia significativa entre ambos conjuntos, puesto que podemos ubicar aquellos que tenían un menor nivel de ruido (5%) presentan una menor pérdida, a diferencia del conjunto perteneciente al ruido con 25%, quienes se ubican en la zona inferior de la figura anteriormente mencionada.

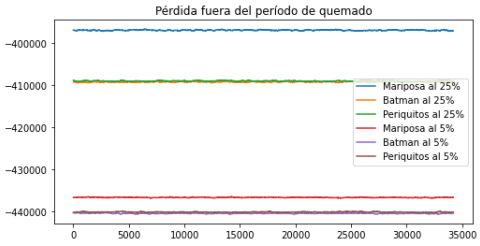

Fig. 11. Finalización en el período de quemado.
Comportamiento de la pérdida una vez finalizado el período de quemado para ambos conjuntos de datos.

A continuación, se presentan de forma más precisa la gráfica correspondiente a la pérdida una vez finalizado el período de quemado para cada caso.

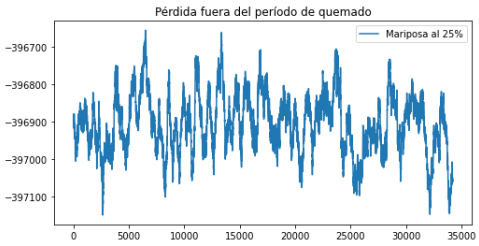

Fig. 12. Finalización en el período de quemado.
Gráfica demostrativa del comportamiento de la pérdida en una vez terminado el período de quemado para la imagen *Mariposa* con un ruido de 25%.

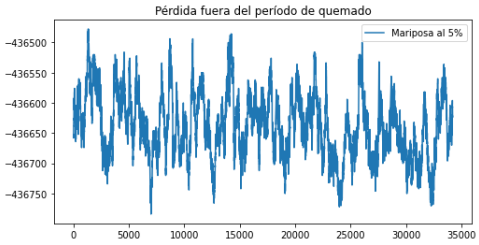

Fig. 13. Finalización en el período de quemado.
Gráfica demostrativa del comportamiento de la pérdida en una vez terminado el período de quemado para la imagen *Mariposa* con un ruido de 5%.

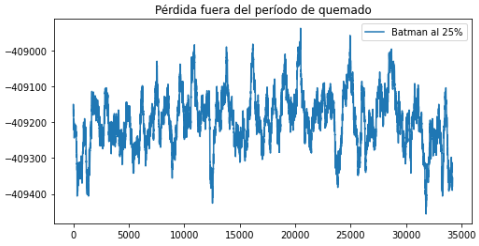

Fig. 14. Finalización en el período de quemado.
Gráfica demostrativa del comportamiento de la pérdida en una vez terminado el período de quemado para la imagen *Batman* con un ruido de 25%.

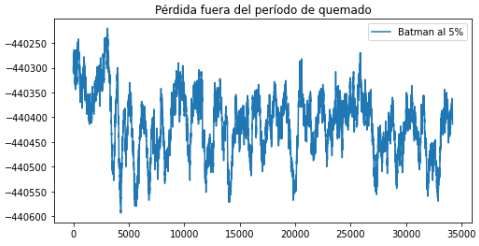

Fig. 15. Finalización en el período de quemado.
Gráfica demostrativa del comportamiento de la pérdida en una vez terminado el período de quemado para la imagen *Batman* con un ruido de 5%.

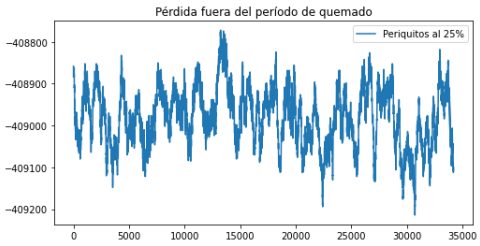

Fig. 16. Finalización en el período de quemado.
Gráfica demostrativa del comportamiento de la pérdida en una vez terminado el período de quemado para la imagen *Periquitos* con un ruido de 25%.

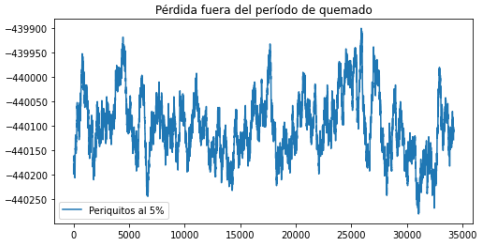

Fig. 17. Finalización en el período de quemado.
Gráfica demostrativa del comportamiento de la pérdida en una vez terminado el período de quemado para la imagen *Periquitos* con un ruido de 5%.

VIII. Conclusiones

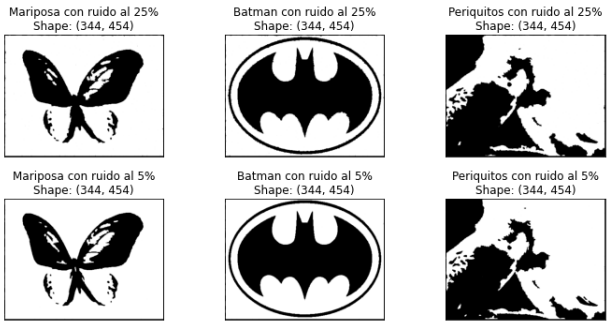

Fig. 18. Resultados.
Comparativo de las imágenes finales, obtenidas después de aplicar nuestro muestreo.

Tal cual podemos apreciar de las imágenes resultantes, podremos notar que realmente la diferencia es mínima para ambos casos. Poniéndonos más "*estrictos*", podremos darnos cuenta que la mejor imagen donde podremos encontrar diferencias (Aunque no tan notorias) es en la de Batman con un ruido al 25%, justo en la parte superior se alcanzan a ver alguno que otro puntito. En la misma imagen, del ala derecha, podremos observar también la misma situación.

Si analizamos las gráficas (En específico la Figura 11), podremos observar que existe una menor pérdida fuera del período de quemado para aquellas imágenes con un ruido del 5%, esto resulta relativamente obvio si pensamos en un mayor *espaciado* entre los pixeles, no hay gran saturación de ruido, por lo que es más probable que el punto que sea clasificado como ruido resulte *aislado* (Sus vecinos cercanos no sean ruido).

Finalmente podremos concluir que si bien es cierto, los resultados no son tan distantes, es preferible tener la menor cantidad de ruido posible en la imagen.

References

[1] Hernández, A. (2017). *Método de Monte Carlo basado en cadenas de Markov*. Junio 15, 2021, de Meta-Learning lab Sitio web: https://mlearninglab.com/2017/11/27/metodo-de-monte-carlo-basado-en-cadenas-de-markov/

[2] Ortiz, M. (2018). *MCMC*. Junio 15, 2021, de ___ Sitio web: https://tereom.github.io/est-computacional-2018/mcmc.html

[3] Stephens, M. (2018). *The Metropolis Hastings Algorithm*. Junio 15, 2021, de ___ Sitio web: https://stephens999.github.io/fiveMinuteStats/MH_intro.html

[4] ___. (___). *Muestreo de Gibbs*. junio 15, 2021, de Wikipedia Sitio web: https://es.wikipedia.org/wiki/Muestreo_de_Gibbs

[5] Ortíz, M. (2018). *Muestreador de gibbs*. Junio 15, 2021, de ___ Sitio web: https://tereom.github.io/est-computacional-2018/muestreador-de-gibbs.html

[6] Causevic, S. (2020). *Reducción de ruido de imagen con muestreo de Gibbs (MCMC)*. Junio 15, 2021, de towards data science Sitio web: https://towardsdatascience.com/image-denoising-with-gibbs-sampling-mcmc-concepts-and-code-implementation-11d42a90e153