UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

ELIMINACIÓN DE RUIDO SPECKLE EN IMÁGENES DE ULTRASONIDO USANDO APRENDIZAJE PROFUNDO

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
DANIEL MENDOZA RODRÍGUEZ

TUTOR:
DRA. JIMENA OLVERES MONTIEL
FACULTAD DE INGENIERÍA

CIUDAD UNIVERSITARIA, CD. MX, MAYO 2023

Resumen

El uso de imágenes de ultrasonido en medicina es una práctica común debido a su naturaleza no invasiva, bajo costo y seguridad en comparación con otros métodos que dependen de la radiación. No obstante, existe una serie de limitaciones, siendo la más común y perjudicial el ruido en la adquisición de las imágenes. La reducción de este ruido es crucial, ya que puede afectar la precisión y eficacia del diagnóstico que se realiza con las imágenes.

En este proyecto se ha analizado la técnica utilizada en investigaciones previas para reducir el ruido *speckle* en imágenes de ultrasonido, se ha encontrado que la mayoría de los enfoques existentes tienden a intentar suprimir por completo el ruido, lo que resulta en imágenes suaves y pierde los detalles estructurales importantes. Por lo tanto, se propone un enfoque diferente que se centra en mejorar la calidad de las imágenes de ultrasonido sin alterar los atributos estructurales y cualitativos, utilizando técnicas de aprendizaje profundo, como son las redes generativas adversarias que se usan ampliamente en el procesamiento de imágenes, obteniendo un resultado superior a comparación de métodos de filtrado clásico.

Abstract

The use of ultrasound images in medicine is a common practice due to its non-invasive nature, low cost, and safety compared to other methods that rely on radiation. However, there are a series of limitations, the most common and harmful being noise in the image acquisition. Reducing this noise is crucial as it can affect the accuracy and effectiveness of the diagnosis made with the images.

In this project, the technique used in previous research to reduce speckle noise in ultrasound images has been analyzed. It has been found that most existing approaches tend to try to completely suppress the noise, resulting in smooth images and losing important structural details. Therefore, a different approach is proposed that focuses on improving the quality of the ultrasound images without altering the structural and qualitative attributes, using deep learning techniques, such as generative adversarial networks that are widely used in image processing, obtaining a superior result compared to classical filtering methods.

Agradecimientos

Expreso mi gratitud a mis padres por brindarme la oportunidad de seguir mi camino académico y enfrentar nuevos desafíos, y a mis hermanos por inspirarme y animarme a cumplir uno de mis sueños. También quiero agradecer a mi universidad, la UNAM, que me ha enseñado a aprovechar al máximo mis habilidades y conocimientos para abordar cualquier desafío.

No podría haber llevado a cabo esta tesis sin el invaluable apoyo de la doctora Jimena Olveres, quien ha actuado como mi tutor y me ha guiado con su conocimiento y comprensión. También estoy agradecido con los profesores de mi programa de maestría en ciencias e ingeniería de la computación de la UNAM, con una mención especial al doctor Boris Escalante, por compartir su sabiduría y estar disponibles en todo momento para ayudarme.

Este trabajo de tesis recibió apoyo de los proyectos UNAM PAPIIT IV100420 y TA101121 y de una beca de nivel maestría del Consejo Nacional de Ciencia y Tecnología.

Índice general

Índice de figuras

Capítulo 1

Introducción

El ultrasonido es una técnica médica que se utiliza para producir imágenes detalladas de las estructuras internas del cuerpo humano. Es una herramienta muy útil para detectar diversos problemas de salud como el dolor, la hinchazón y las infecciones en los órganos internos. El proceso de realizar un ultrasonido es fácil y no requiere una preparación especial. Además, es una opción segura y no invasiva que no utiliza radiación. Esto lo convierte en una herramienta muy valiosa para el diagnóstico de diversos problemas de salud.

La calidad de las imágenes obtenidas con ultrasonido puede verse afectada por la presencia de ruido, que depende de la señal. Este ruido se conoce como ruido *speckle* (el cual es de naturaleza multiplicativa) y es una propiedad intrínseca de la tecnología de ultrasonido. El ruido *speckle* reduce la resolución y el contraste de las imágenes, lo que puede afectar la precisión diagnóstica. Por esta razón, es fundamental llevar a cabo una limpieza de las imágenes para reducir el ruido, manteniendo al mismo tiempo las características importantes de la imagen. Esto mejorará la precisión de los diagnósticos y aumentará el valor clínico de las imágenes por ultrasonido.

Pix2Pix es una red neuronal de aprendizaje profundo que se expone en un trabajo publicado en el área de procesamiento de imágenes y visión por computadora. Pix2Pix puede resolver muchos problemas de procesamiento de imágenes que requieran la "traducción" de una imagen original (la entrada) a una nueva imagen ya procesada (la salida). Esta red puede aplicarse a una amplia gama de tareas, incluyendo la síntesis

de imágenes a partir de mapas de segmentos, la generación de imágenes a color a partir de imágenes en blanco y negro, la conversión de imágenes de mapas topográficos a imágenes de fotografías aéreas, la transformación de bocetos en imágenes realistas e incluso la tarea de completar imágenes con espacios vacíos como se puede ver en la Figura 1.1 de esta última tarea se toma la idea de usar esta arquitectura para la eliminación de ruido y reconstruir la imagen.

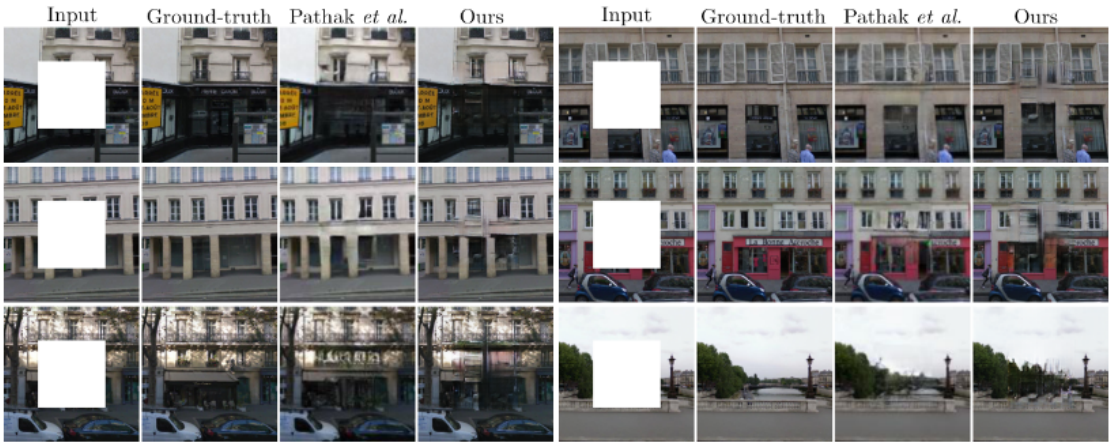

Figura 1.1: Tarea de completar los espacios vacíos en una imagen, realizada por una arquitectura del tipo Pix2Pix, imagen tomada de Phillip Isola (2018).

Pix2Pix es una red del tipo generativa antagónica, esto quiere decir que la red cuenta con dos módulos, un generador, el encargado de producir una imagen, en este caso una imagen de ultrasonido limpia, y un discriminador, el cual se encarga de determinar qué tan bien se generó la imagen. La competencia entre estos dos módulos es lo que le da el nombre de antagónica a la red. Esta red tiene dos principales cambios que se proponen que difieren con una GAN tradicional que son: el uso de una entrada condicional, en este caso es un elemento fundamental, ya que esta entrada corresponderá a la imagen de ultrasonido a la cual se le desea eliminar el ruido y el uso de *patch* que divide la imagen en sectores, para evaluar la similitud de la imagen, esto sirve para evaluar por ventanas la correspondencia entre la imagen generada y la objetivo.

En el artículo se especifica de una forma clara cómo reproducir el modelo, este consta de dos partes, el generador, siendo una U-Net con entrada condicional y un

discriminador siendo una arquitectura del tipo PatchGAN, las especificaciones en detalle se pueden ver en la sección 6.1 de [Phillip Isola (2018)].

Figura 1.2: Ultrasonido portátil, imagen tomada de MedImaging (2019).

1.1. Problema

El uso de imágenes por ultrasonido en el diagnóstico médico es ampliamente reconocido y utilizado debido a su naturaleza no invasiva, bajo costo y facilidad de implementación como se ve en la Figura 1.2 donde se puede apreciar una máquina de ultrasonido portátil. Sin embargo, la adquisición de estas imágenes a menudo está acompañada de ruido, específicamente el ruido *speckle*, que es una propiedad inherente de la ecografía médica. Este tipo de ruido tiene una naturaleza multiplicativa y puede reducir la resolución y el contraste de la imagen. La reducción del ruido es crucial, ya que el ruido puede limitar la precisión del diagnóstico. Por lo tanto, es importante eliminar el ruido sin afectar las características importantes de la imagen.

1.2. Objetivo

Este trabajo propone desarrollar una solución efectiva y eficiente para mejorar la calidad de las imágenes de ultrasonido mediante el uso de una red neuronal, específi-

camente la red Pix2Pix. Mientras que la mayoría de los métodos existentes se enfocan en intentar eliminar la mayor parte del ruido *speckle*, lo que resulta en imágenes poco claras y sin detalles relevantes, el enfoque propuesto busca reducir el ruido *speckle* de manera cuidadosa, manteniendo los aspectos estructurales y cualitativos importantes de la imagen original. Este enfoque permitirá una evaluación más precisa y detallada de la imagen de ultrasonido para fines médicos.

1.3. Organización de la tesis

La tesis se estructura en seis capítulos. En el capítulo 1 Introducción, se menciona la importancia de tratar la problemática relacionada con la reducción de ruido en imágenes de ultrasonido y el objetivo planteado para la tesis. En el capítulo 2 Conceptos generales, se abarcan los conceptos básicos necesarios para el proyecto, pero sin profundizar en ellos. El capítulo 3 Estado del arte: reducción de ruido *speckle*, presenta una revisión de los métodos más utilizados para reducir el ruido en imágenes de ultrasonido, divididos en dos grandes categorías: métodos clásicos como filtros y métodos modernos basados en aprendizaje profundo. La metodología se explicada en el capítulo 4 Redes neuronales artificiales para la reducción de ruido *speckle*, en donde se describen los procesos y conceptos utilizados para el desarrollo de la herramienta. En el capítulo 5 Desarrollo, se detalla la implementación de los conceptos vistos en la metodología para el desarrollo de la herramienta. Finalmente, en el capítulo 6 Resultados y conclusiones, se comparan y discuten los resultados obtenidos, así como se plantean futuras mejoras e implementaciones que se pueden realizar.

Capítulo 2

Conceptos generales

Para llevar a cabo el proyecto se necesitaba una comprensión de cómo funciona el aprendizaje profundo, con énfasis en redes neuronales artificiales, y el ultrasonido, así como el procedimiento que se realiza para la adquisición de dichas imágenes. A continuación, se expondrá de forma concisa una pequeña introducción al fenómeno de ultrasonido y al concepto de aprendizaje profundo, para poder comprender sus respectivos funcionamientos.

2.1. Ultrasonido

El ultrasonido es una onda de sonido con una frecuencia que supera los 20 kHz. Transporta energía y se propaga a través de varios medios como una onda de presión pulsante. Se describe mediante una serie de parámetros de onda, como la densidad de presión, la dirección de propagación y el desplazamiento de partículas. Si el desplazamiento de la partícula es paralelo a la dirección de propagación, entonces la onda se denomina onda longitudinal o de compresión. Si el desplazamiento de la partícula es perpendicular a la dirección de propagación, se trata de una onda transversal o de corte. La interacción de las ondas de ultrasonido con el tejido está sujeta a las leyes de la óptica geométrica. Incluye reflexión, refracción, dispersión, difracción, interferencia y absorción. Excepto por la interferencia, todas las demás interacciones reducen la intensidad del haz de ultrasonido, como se puede ver en [S. B. P.S. Hiremath Prema

T. Akkasaligar (2013)].

Las imágenes generadas a través de ultrasonido también se conocen como ecografías. Este proceso involucra el uso de una pequeña sonda llamada transductor y un gel que se aplica directamente sobre la piel. Las ondas sonoras de alta frecuencia se transmiten desde la sonda a través del gel y dentro del cuerpo, y luego la sonda recoge los sonidos que rebotan. Una computadora utiliza estos sonidos para crear una imagen. Se utilizan cuatro métodos diferentes de ultrasonido en imágenes médicas como se ve en [Carovac A (2011)].

- Modo A: El Modo A es el tipo más básico de ultrasonido. Consiste en el escaneo de un solo transductor a través de una línea del cuerpo, con los ecos que se muestran en la pantalla en función de su profundidad. Además, está enfocado en detección de un tumor o cálculo específico.

- Modo B: En el ultrasonido en modo B, una matriz lineal de transductores escanea simultáneamente un plano a través del cuerpo que se puede ver como una imagen bidimensional en la pantalla. Para el proyecto se usó este tipo de ultrasonido que se puede ver en la Figura 2.1

- Modo M: En el Modo M, una serie de exploraciones en Modo B realizadas de manera rápida y sucesiva permiten a los médicos observar y medir el movimiento, ya que las imágenes se muestran en secuencia en la pantalla. Esto permite ver cómo los límites de los órganos que producen reflejos cambian en relación con la sonda.

- Modo Doopler: El Modo Doppler se utiliza para medir y visualizar el flujo sanguíneo, utilizando el efecto Doppler. El ultrasonido Doppler es importante en medicina, ya que mejora mediciones que evalúan si las estructuras (generalmente la sangre) se están moviendo hacia la sonda o en sentido contrario y su velocidad relativa.

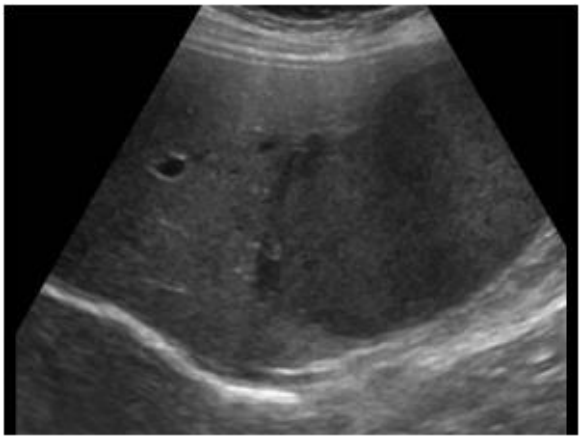

Figura 2.1: Imagen de ultrasonido modo B, imagen tomada de Al-Dhabyani W (2020).

2.1.1. Ruido *speckle*

En las imágenes de ultrasonido, el contenido de ruido es multiplicativo y no gaussiano. Generalmente, dicho ruido es más difícil de eliminar que el ruido aditivo, porque la intensidad del ruido varía con la intensidad de la imagen. Un modelo de ruido multiplicativo viene dado por la Ecuación 2.1.

$$y_{ij} = X_{ij} n_i \tag{2.1}$$

donde la imagen con ruido es y_{ij} es el producto de la imagen original X_{ij} y el ruido no gaussiano n_i. Los índices i, j representan la posición espacial sobre la imagen. En la mayoría de las aplicaciones que implican ruido multiplicativo, se supone que el contenido de ruido es estacionario con media unitaria y varianza de ruido desconocida σ^2.

El ruido *speckle* se representa por una distribución de Rayleigh, donde la distribución de la magnitud de un vector cuyas componentes X e Y son variables aleatorias normales. Así como la distribución normal tiene dos parámetros que la determinan completamente: la media μ y la desviación estándar σ de la muestra; la distribución de Rayleigh está determinada por un único parámetro σ, como se puede ver en [S. B. P.S. Hiremath Prema T. Akkasaligar (2013)]. Se puede observar la forma de este ruido en Figura 2.2

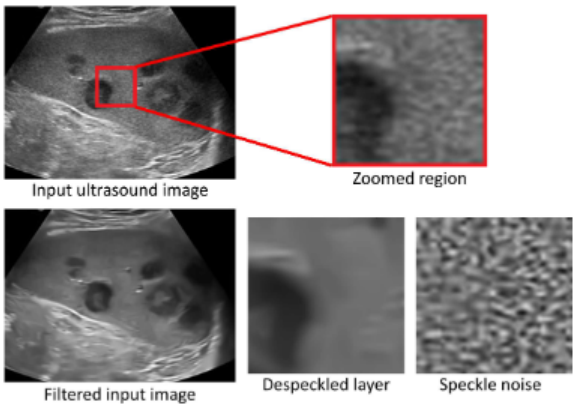

Figura 2.2: Ruido *speckle* en imágenes de ultrasonido, imagen tomada de Lei Zhu (s.f.).

2.1.2. Generación de imágenes de ultrasonido

El transductor de ultrasonido genera ondas de ultrasonido y es sostenido con una mano para ajustar su posición y ángulo para enviar las ondas a través de las estructuras a ser visualizadas. Las ondas de ultrasonido son emitidas rápidamente desde el transductor y viajan a través de los tejidos y fluidos. Algunas de las ondas son reflejadas de vuelta al transductor y, al analizar estas ondas reflejadas, la máquina de ultrasonido crea una imagen de los tejidos.

Las ondas de ultrasonido son generadas por cristales cerámicos con propiedades piezoeléctricas, estos cristales están conectados al frente del transductor. Los cristales pueden convertir corrientes eléctricas en ondas de ultrasonido y viceversa, permitiendo la interpretación y traducción de la señal eléctrica en una imagen por parte de la máquina de ultrasonido. Las ondas de ultrasonido son enviadas en pulsos, y cada pulso consiste en varias ondas emitidas en 1 a 2 milisegundos. Las ondas reflejadas tienen la misma velocidad que las ondas emitidas, pero su amplitud, frecuencia y ángulo de incidencia pueden ser diferentes, y la máquina de ultrasonido utiliza estas variaciones para crear una imagen del tejido.

2.2. Aprendizaje de máquina

El aprendizaje de máquina es una rama de la inteligencia artificial que se centra en crear sistemas que aprendan sin ser expresamente programadas para ello, en función

de los datos que consumen. Este campo de estudio forma parte de la informática y se interesa por dotar a las máquinas de la habilidad de aprender de forma autónoma. El aprendizaje se entiende como la capacidad de la máquina para generalizar el conocimiento adquirido a través de experiencias previas.

El aprendizaje se divide en tres distintos paradigmas, cada uno de ellos se enfoca en una metodología diferente para lograr el aprendizaje automático. Estos tres enfoques se conocen como aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

- Aprendizaje supervisado, que es el tipo de aprendizaje que se basa en descubrir la relación entre los datos de entrada y los datos de salida, y es el empleado en este trabajo.

- Aprendizaje no supervisado, es un aprendizaje donde se consigue generar conocimiento únicamente a partir de los datos de entrada.

- Aprendizaje por refuerzo, este aprendizaje se basa en castigar y recompensar las acciones de un agente, para que este mismo determine las acciones con mayor recompensa.

Existen métodos de aprendizaje profundo que utilizan redes neuronales artificiales para aprender patrones y estructuras complejas en los datos. Este método se basa en la idea de que la información se aprende de manera incremental y jerarquizada, lo que significa que las características más simples se aprenden primero y luego se utilizan para construir características más complejas. Por ejemplo, una red neuronal puede comenzar aprendiendo qué es un ojo, una nariz, una boca y una oreja antes de aprender qué es una cara. La cantidad de aprendizaje y la complejidad de las características que una red neuronal puede aprender depende de la cantidad de capas que tenga. Mientras más capas tenga, más complejas serán las características que pueda aprender.

2.2.1. Estructura de una Red Neuronal Artificial

Las RNA (Red Neuronal Artificial) están compuestas principalmente por neuronas, estas toman entradas numéricas, las cuales son procesadas para generar una salida numérica. La neurona es una suma ponderada de las entradas más el sesgo y pasado por una función de activación, se representa por la Ecuación 2.2

$$Y = \varphi(b + W_1X_1 + W_2X_2 + ... + W_mX_m) \tag{2.2}$$

donde φ es la función de activación, b es el sesgo, W es el peso asignado a cada una de las entradas y X son los valores de entrada de cada una de las neuronas.

Se puede apreciar en la ecuación que si se omite la función de activación φ se asemeja a la ecuación de una recta (esto se puede lograr haciendo que la función de activación sea la función identidad), como se puede ver en [Bonifacio Martín del Brío (2007)]. La función de activación φ, tiene su importancia al crear las redes neuronales artificiales, ya que sin ellas se tendría problemas con la linealidad de las neuronas, haciendo que al sumar rectas el resultado sea otra única recta, que sería lo equivalente a tener una única neurona, para ello estas funciones de activación hacen que se pierda la linealidad en las neuronas, algunas de las funciones de activación más comunes se pueden ver en la Figura 2.3.

Al momento de juntar neuronas estamos creando una red, las neuronas las podemos juntar en filas y columnas, cuando las agrupamos en columna se dice que la capa se hace más grande, mientras que si las juntamos en filas estamos creando más capas. La red se agrupa en distintas capas, las cuales se pueden clasificar en tres distintos tipos:

- Capa de entrada, es la entrada a la red, esta contiene la información a procesar.
- Capas ocultas, las cuales son las encargadas de procesar la información, generalmente estas capas suelen ser más de una, y la forma más habitual de usarlas es conectar todas y cada una de las neuronas con todas y cada una de las neuronas

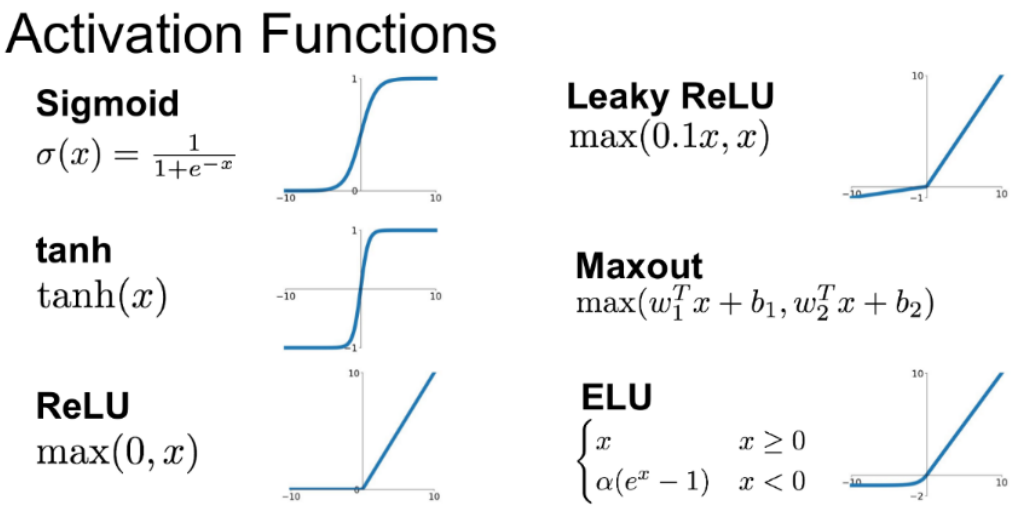

Figura 2.3: Funciones de activación comunes, imagen tomada de KeepCoding (2022).

de las capas adyacentes.

- Capa de salida, en donde se obtiene el resultado final de la red.

Se puede ver un esquema general de una RNA y neurona en la Figura 2.4.

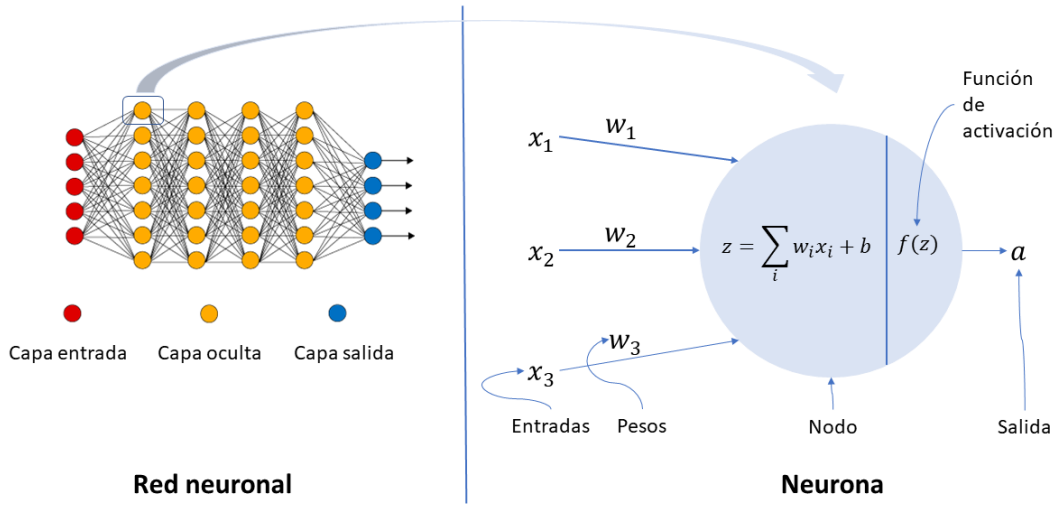

Figura 2.4: Representación de una red neuronal artificial y su neurona, imagen tomada de Ponce (2021).

2.2.2. Aprendizaje de una Red Neuronal Artificial

Con las RNA podemos crear un modelo capaz de resolver problemas complejos, lo único que tenemos que hacer una vez creada nuestra red es ajustar los parámetros, específicamente los pesos W. Hacer esto manualmente es una locura, ya que dependiendo del tamaño del modelo, hay que asignarle un valor a miles de variables.

Para asignarle los valores a las variables W, se usa el descenso por gradiente representado en la Figura 2.5, el cual busca el punto mínimo de la función de coste (optimiza los resultados obtenidos). Para realizar esta operación es necesario calcular las derivadas parciales de los parámetros y calcular el gradiente. Una vez obtenido se avanza cierta cantidad (conocida como tasa de aprendizaje) en sentido contrario al gradiente. Este procedimiento se repite muchas veces hasta minimizar el error.

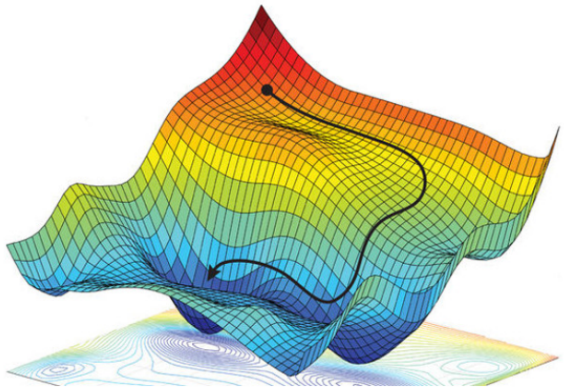

Figura 2.5: Representación del descenso del gradiente, imagen tomada de admin (s.f.).

Para poder encontrar las derivadas parciales que se necesitan para calcular el gradiente se necesita conocer el error proporcionado para cada neurona. Al momento de procesar los datos y querer obtener un resultado con una RNA, estos se procesan de izquierda a derecha, de la capa de entrada a la capa de salida, sin embargo, para detectar el porcentaje de error de cada neurona se hace a la inversa de derecha a izquierda.

Este algoritmo conocido como *backpropagation* se basa en comparar el error que hay entre la entrada y la salida esperada. Una vez calculado el porcentaje de error se comienza a recorrer de derecha a izquierda neurona por neurona verificando cuánto

aportó en el error final. Se realiza en este sentido porque el error crece hacia la derecha, así que si el error es poco en una neurona las capas anteriores a esta con las cuales este conectadas serán aún menos, podemos ver el pseudocódigo en el Algoritmo 1 y una explicación más detallada en [Guo, Nguyen, Vu, y Bui (2021)].

Algorithm 1 Algoritmo de *Backpropagation*

proceso de entrenamiento
$X \leftarrow$ conjunto de datos de entrenamiento de tamaño mxn
$y \leftarrow$ etiquetas en X
$w \leftarrow$ pesos de las capas
$l \leftarrow$ numero de capas en la red neuronal
$D_{ij}^{(l)} \leftarrow$ el error para todo l, i, j
$t_{ij}^{(l)} \leftarrow 0$ para todo l, i, j
for $i = 1$ hasta m **do**
    $a^{(l)} \leftarrow$ propagación hacia adelante $(x^{(i)}, w)$
    $d^{(l)} \leftarrow a(L) - y(i)$
    $t_{ij}^{(l)} \leftarrow t_{ij}^{(l)} + a_j^{(l)} \cdot t_i^{(l+1)}$
if $j \neq 0$ **then**
    $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)} + \lambda w_{ij}^{l}$
else
    $D_{ij}^{(l)} \leftarrow \frac{1}{m} t_{ij}^{(l)}$
    where $\frac{\delta}{\delta w_{ij}^{l}} J(w) = D_{ij}^{(l)}$

Capítulo 3

Estado del arte: reducción de ruido *speckle*

El ruido en imágenes es uno de los efectos menos deseados y más susceptibles. Una imagen con ruido puede ser tratada por distintos medios, por lo general se podrá dividir en dos grandes grupos, métodos clásicos, como son el uso de filtros y métodos modernos que en los últimos años han tomado mucha relevancia gracias a los resultados obtenidos, siendo estos basados en el uso del aprendizaje profundo.

3.1. Reducción de ruido *speckle* por medio de filtros

Los filtros de reducción de ruido *speckle* se originan en la investigación de la comunidad dedicada al estudio de radares de apertura sintética. Estos filtros se aplican posteriormente a las imágenes de ultrasonido a principios de la década de 1980. Hay dos clasificaciones principales de filtros de reducción, filtros espaciales de escala única y filtros multiescala de dominio transformado. El filtro espacial actúa sobre una imagen suavizándola; es decir, reduce la variación de intensidad entre píxeles adyacentes. El filtro espacial de ventana deslizante simple reemplaza el valor central en la ventana con el promedio de todos los valores de píxeles vecinos, incluido él mismo. Al hacer esto, reemplaza píxeles que no son representativos de su entorno. Se implementa con una máscara de convolución, que proporciona un resultado que es una suma ponderada

de los valores de un píxel y sus vecinos. También se le llama filtro lineal. La máscara o núcleo es una matriz cuadrada, a menudo se utiliza un núcleo de 3×3. Si los coeficientes de la máscara suman uno, entonces el brillo promedio de la imagen no cambia. Si los coeficientes suman cero, el brillo promedio se pierde y devuelve una imagen oscura como se ve en [Erick Cuevas (2010)].

Entre filtros comunes para la reducción de ruido *speckle* podemos encontrar:

3.1.1. Filtro Lee

Se basa en el enfoque de que el suavizado se realiza en el área que tiene una varianza baja. Sin embargo, el suavizado no se realizará en el área de alta varianza. El filtro de Lee asume que la imagen se puede aproximar mediante un modelo lineal. La principal desventaja del filtro Lee es que tiende a ignorar el ruido en las áreas más cercanas a los bordes y las líneas, el filtro se representa por la Ecuación 3.1.

$$Y_{ij} = \bar{K} + W * (C - \bar{K}) \tag{3.1}$$

donde Y_{ij} es el valor de la escala de grises del píxel en (i, j) después del filtrado. Si no hay suavizado, el filtro generará solo el valor de intensidad media $\bar{K}$ del *kernel* K, de lo contrario la diferencia entre el píxel central C y $\bar{K}$ se calcula y se multiplica con una función de ponderación W dada en la Ecuación 3.2.

$$W = \frac{\sigma_k^2}{\sigma_k^2 + \sigma^2} \tag{3.2}$$

Luego se suma con $\bar{K}$, donde σ_k^2 es la varianza de los valores de píxel dentro del *kernel* dada por la Ecuación 3.3:

$$\sigma_k^2 = \frac{1}{M^2} * \Sigma_{u,v=0}^{M-1}(K_{uv} - \bar{K})^2 \tag{3.3}$$

donde $M \times M$ es el tamaño del *kernel* y K_{uv} es el valor de píxel dentro del *kernel*

en los índices u y v, $\bar{K}$ es el valor medio de intensidad del *kernel*. El parámetro σ^2 es la varianza de la imagen X, como se puede ver en [P. T. A. P.S. Hiremath y Badiger (2013)].

3.1.2. Filtro Kuan

El filtro Kuan es un método de reducción de ruido que sigue un proceso de filtrado similar al filtro Lee. Este método aplica un filtro espacial a cada píxel de una imagen y utiliza las estadísticas locales de los valores de los píxeles vecinos para filtrar el valor del píxel central del *kernel*. El filtro Kuan utiliza un valor de suavizado para controlar la cantidad de reducción de ruido y estima la varianza del ruido para aplicar el filtro de manera adecuada. El filtro se representa por la Ecuación 3.4.

$$W = \frac{1 - C_u/C_i}{1 + C_u} \tag{3.4}$$

La función de ponderación se calcula a partir del coeficiente de variación de ruido estimado de la imagen, C_u dado por la Ecuación 3.5.

$$C_u = (ENL)^{-\frac{1}{2}} \tag{3.5}$$

Y el coeficiente de variación C_i de la imagen dado por la Ecuación 3.6.

$$C_i = \frac{\sigma_k}{\bar{K}} \tag{3.6}$$

donde ENL es dada por la Ecuación 3.7.

$$ENL = (\frac{\bar{K}}{\sigma_k})^2 \tag{3.7}$$

Como se puede ver en [P. T. A. P.S. Hiremath y Badiger (2013)].

3.1.3. Filtro Frost

El filtro Frost es un método de reducción de ruido que preserva las características importantes de la imagen en los bordes, utilizando un filtro simétrico circular. Este filtro utiliza una ventana de filtro individual y aplica una función exponencialmente decreciente para reducir el ruido en la imagen. El filtro Frost requiere un factor de vaciado para ajustar la cantidad de reducción de ruido, representado en la Ecuación 3.8.

$$Y = \frac{\Sigma(x * W)}{\Sigma W} \tag{3.8}$$

donde x es el valor del *kernel* de $n \times n$ encargado de recorrer la imagen, Y es el valor del filtro y W es el peso asignado a cada píxel, como se puede ver en [P. T. A. P.S. Hiremath y Badiger (2013)].

3.1.4. Filtro Mediana

El filtro de la mediana es un tipo de filtro estadístico no lineal que se basa en la mediana de los datos de una muestra para su funcionamiento. En estadística, la mediana es el valor de una variable que deja el mismo número de datos por encima y por debajo de él. Por lo tanto, los conjuntos de datos iguales o menores que la mediana representan el 50 % de los datos de la muestra, mientras que los conjuntos de datos mayores que la mediana representan el otro 50 %. El filtro de la mediana utiliza esta propiedad estadística para eliminar el ruido de una imagen, reemplazando cada píxel con la mediana de los valores de los píxeles vecinos en una ventana determinada.

Considerando que $x_1, x_2 ... x_n$ son los datos de una muestra ordenada en orden creciente, la mediana quedaría definida como la Ecuación 3.9.

$$M_e = X_{\frac{n+1}{2}} \tag{3.9}$$

Una vez que los valores han sido ordenados en orden creciente o decreciente, y

si n es impar será M_e la observación central de estos, si n es par será el promedio aritmético de las dos observaciones centrales, esto se representa en la Ecuación 3.10.

$$M_e = \frac{X_{\frac{n}{2}} + X_{\frac{n+1}{2}}}{2} \tag{3.10}$$

El filtro de la mediana reemplaza cada píxel de una imagen con la mediana de los valores de intensidad de los píxeles en una región específica alrededor del píxel en cuestión. Es decir, en lugar de utilizar el valor del propio píxel, el filtro utiliza la mediana de los valores de intensidad de los píxeles vecinos dentro de una determinada región de influencia $R(x.y)$ definida por la Ecuación 3.11 del filtro, para reducir el ruido en la imagen.

$$\acute{I}(x, y) = M_e(R(x, y)) \tag{3.11}$$

Como se puede ver en [Erick Cuevas (2010)].

3.1.5. Filtro Fastnl

El filtro Fastnl reemplaza el valor de un píxel por un promedio de una selección de otros valores de píxeles, los parches pequeños centrados en los otros píxeles se comparan con la ventana centrada en el píxel de interés, y el promedio se realiza solo para los píxeles que tienen la ventana cercana a la ventana actual.

La eliminación de ruido de una imagen en color $u = (u1, u2, u3)$ y un cierto parche $B = B(p, f)$ (centrado en p y con tamaño $(2f + 1) \times (2f + 1)$).

$$\hat{B}_i = \frac{1}{C(p)} \Sigma_{Q-Q(q,f)\epsilon B(p,r)} u_i(Q) w(B, Q) \tag{3.12}$$

$$C = \Sigma_{Q-Q(q,f)\epsilon B(p,r)} w(B, Q) \tag{3.13}$$

donde $i = 1, 2, 3$, $B(p, r)$ indica una vecindad centrada en p y con tamaño $(2r + 1) \times (2r + 1)$ píxeles y $w(B(p, f), B(q, f))$ corresponde a la Ecuación 3.14.

$$w(p,q) = \exp^{-\frac{max(d^2-2\sigma^2,0.0)}{h^2}} \tag{3.14}$$

De esta forma, aplicando el procedimiento para todos los parches de la imagen, dispondremos de $N^2 = (2f+1)^2$ posibles estimaciones para cada píxel. Estas estimaciones se pueden promediar finalmente en cada ubicación de píxel.

$$\hat{u}_i(p) = \frac{1}{N^2}\Sigma_{Q-Q(q,f)|q\epsilon B(p,f)}\hat{Q}_i(p) \tag{3.15}$$

Como se puede ver en [Antoni Buades (2015)].

3.2. Reducción de ruido *speckle* por medio de aprendizaje profundo

Es imposible construir un filtro clásico que pueda quitar todas las imperfecciones o ruidos de la imagen y al mismo tiempo mantener intactas estructuras de la imagen consideradas como importantes, por ello se ha optado por usar otros métodos al momento de eliminar el ruido con la esperanza de que se tengan mejores resultados, entre estos métodos se ha usado el aprendizaje profundo.

En el proceso de reducción de ruido en imágenes utilizando técnicas de aprendizaje profundo, se suele utilizar un tipo especial de red neuronal artificial conocido como autocodificador. El autocodificador usado para la eliminación de ruido en imágenes se caracteriza por tener la entrada y la salida iguales, entra una imagen sale una imagen. A diferencia de las redes neuronales convencionale, los autocodificadores utilizan un codificador en sus capas intermedias para reducir el tamaño de la imagen y luego un decodificador para aumentarla nuevamente a su tamaño original de manera simétrica.

La arquitectura del autocodificador permite que la capa central sea una representación comprimida de las variables de entrada. En este sentido, las capas del codificador crean una versión reducida de la imagen original, mientras que las capas del decodificador intentan recuperar la imagen original a partir de su versión comprimida. Al aprender a reducir la imagen original a una versión comprimida y luego restaurar-

la a su forma original, el autocodificador puede aprender a eliminar el ruido de las imágenes de manera efectiva.

Gracias a la arquitectura del autocodificador, se puede crear una versión reducida de la imagen original mediante el codificador, y luego restaurarla a su forma original mediante el decodificador, permitiendo así que el modelo aprenda a reducir el ruido de las imágenes, se puede ver la arquitectura en la Figura 3.1.

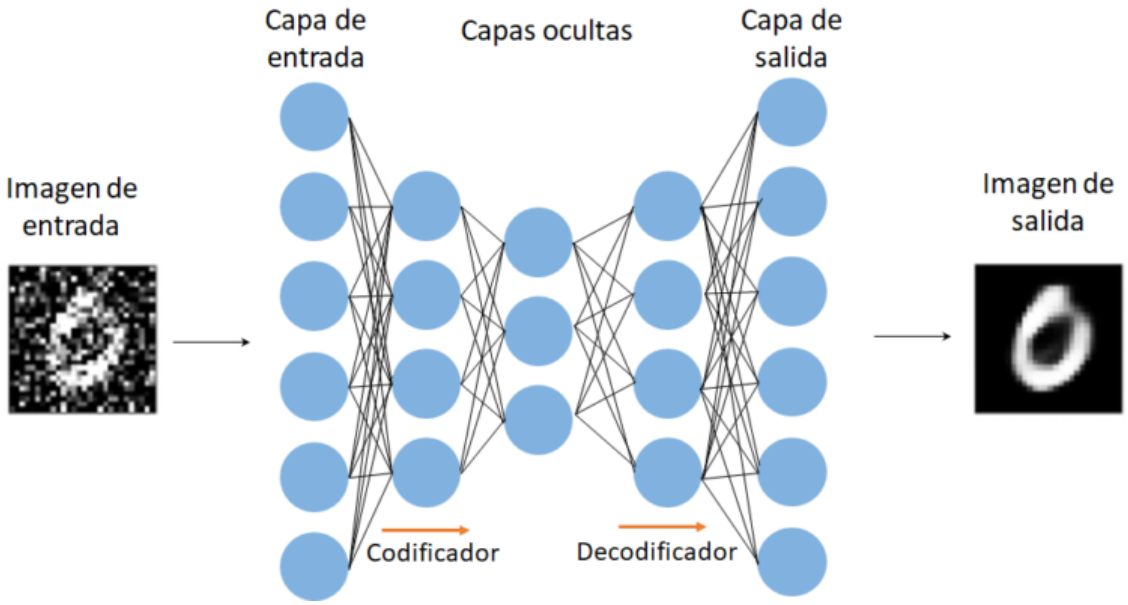

Figura 3.1: Arquitectura general de un autocodificador para eliminar ruido de imágenes, imagen tomada de Esteves (2020).

3.2.1. Trabajos anteriores de reducción de ruido *speckle* por medio de aprendizaje profundo

Existen trabajos sobre la reducción del ruido *speckle* en imágenes de ultrasonido médicas usando aprendizaje profundo, todas ellas centrándose en la eliminación del ruido y dejando de lado lo que es la conservación de la estructura. En todas ellas se toma como idea central el autocodificador implementando modificaciones sobre la estructura de este. Entre ellos podemos encontrar trabajos como [Onur Karaoglu (2021)] donde se puede apreciar la arquitectura de la Figura 3.2.

También podemos encontrar en otros trabajos arquitecturas como son las GAN

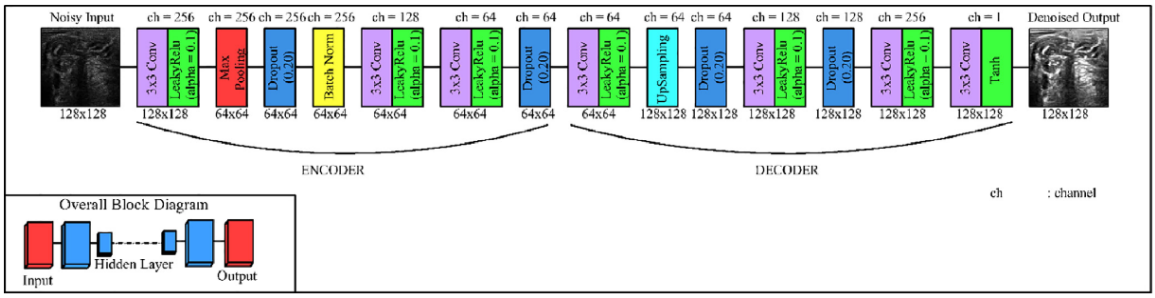

Figura 3.2: Arquitectura tipo autocodificador, imagen tomada de Onur Karaoglu (2021).

(por sus siglas en inglés, Generative Adversarial Networks), la cual tiene como principal característica generación de imágenes, de este tipo de arquitectura podemos encontrar trabajos como [Dietrichson y cols. (2018)]. Donde se pueden ver arquitecturas del tipo GAN que se muestra en la Figura 3.3.

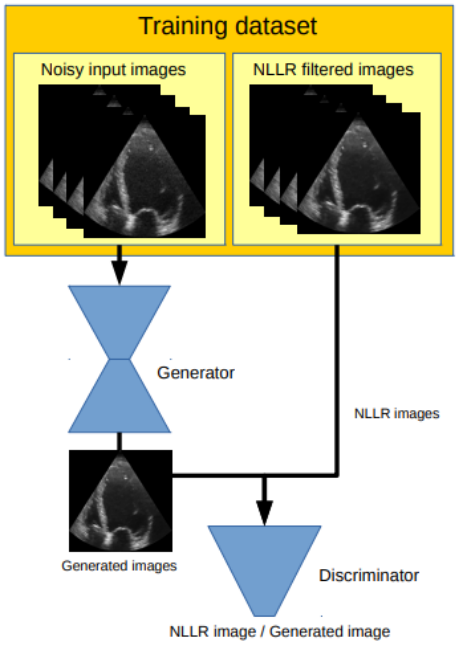

Figura 3.3: Arquitectura tipo GAN usada para limpiar imágenes de ultrasonido, imagen tomada de Dietrichson y cols. (2018).

Existen algunos cuantos trabajos parecidos al ya mencionado, sin embargo casi ninguno le da importancia a la conservación de la estructura de la imagen y aplicación sobre imágenes de ultrasonido.

Capítulo 4

Redes neuronales artificiales para la reducción de ruido *speckle*

4.1. Redes antagónica generativas

Las redes antagónicas generativas son una forma inteligente de entrenar un modelo de aprendizaje no supervisado. Se trata de un modelo generativo que se basa en el aprendizaje profundo y que son capaces de generar imágenes a partir de un conjunto de datos determinado, las imágenes generadas imitan las características de las imágenes del conjunto de datos. Estas redes son caracterizadas por componerse de dos partes, el generador y el discriminador.

4.1.1. Pix2Pix

Pix2Pix es una red del tipo generativa antagónica, la arquitectura del la red esta propuesta en el *paper* [Phillip Isola (2018)], esto quiere decir que la red cuenta con dos módulos, un generador, el encargado de producir una imagen, en este caso una imagen de ultrasonido limpia, y un discriminador, el cual se encarga de determinar que tan bien se esta generando la imagen.

En el artículo se especifica de una forma clara cómo reproducir el modelo, este consta de dos partes, el generador, siendo una U-Net con entrada condicional, y un

discriminador, siendo una arquitectura del tipo PatchGAN, las especificaciones en detalle se pueden ver en la sección 6.1 del *paper* [Phillip Isola (2018)], aquí solo se dará los elementos para reproducir la arquitectura básica del *paper*.

Para el generador, que está compuesto de un codificador y un decodificador, se establece que cada bloque del codificador será dada por [Convolución - Normalización de lote - Leaky ReLU], donde la convolución se aplica a la imagen, con el fin de comprimir su información, la normalización de lote es una técnica que ayuda al entrenamiento, al añadir un paso extra entre la neurona y la función de activación, con el fin de normalizar las funciones de salida, y la función de activación de Leaky ReLU ayuda a evitar que la función se sature en 0, ya que tiene una pendiente pequeña a diferencia de la ReLU.

Mientras que el decodificador tiene [Convolución transpuesta - Normalización de lote - Abandono (aplicado a los primeros 3 bloques) - ReLU], donde la convolución transpuesta sirve para descomprimir los datos de la imagen, la normalización del lote es la misma que en el bloque de codificador, el abandono al igual que el la normalización de lote es una técnica que ayuda al entrenamiento, consiste en desactivar un porcentaje aleatorio de neuronas, y por ultimo una función de activación del tipo ReLU.

Los bloques de codificador y decodificador se conectan entre sí con conexiones de salto, siendo 8 bloques del codificador con 64, 128, 256, 512, 512, 512, 512, 512, filtros respectivamente y el decodificador de 7 bloques con un número de filtros de 512, 512, 512, 512, 256, 128, 64, respectivamente para cada bloque. La arquitectura se puede ver en la Figura 4.1.

Para entrenarlo se calcula la pérdida del generador y se le suma λ con valor de 100 definido en el *paper* y se multiplica por la pérdida L1 entre la imagen generada y la imagen objetivo. Como se muestra en la Figura 4.2.

El discriminador es una red convolucional PatchGAN que clasifica si cada parche de imagen es real o no real, cada bloque del discriminador está dado por [Convolución - Normalización de lote - Leaky ReLU], este recibe dos entradas que se concatenan, la imagen de entrada y la imagen objetivo (que debe clasificar como real) y la imagen de

entrada y la imagen generada (que debe clasificar como falsa), el número de bloques descritos en el artículo son 4 bloques con 64, 128, 256, 512 filtros. Arquitectura vista en la Figura 4.3.

Para entrenar se calcula la pérdida de la imagen generada y la imagen real, las cuales se suman para generar la pérdida total. Como se muestra en la Figura 4.4.

Para ambas redes se usa el algoritmo ADAM para la optimización, con una tasa de aprendizaje de 0.0002, una B1=0.5 y B2=0.999 como se ve en el *paper* [Phillip Isola (2018)].

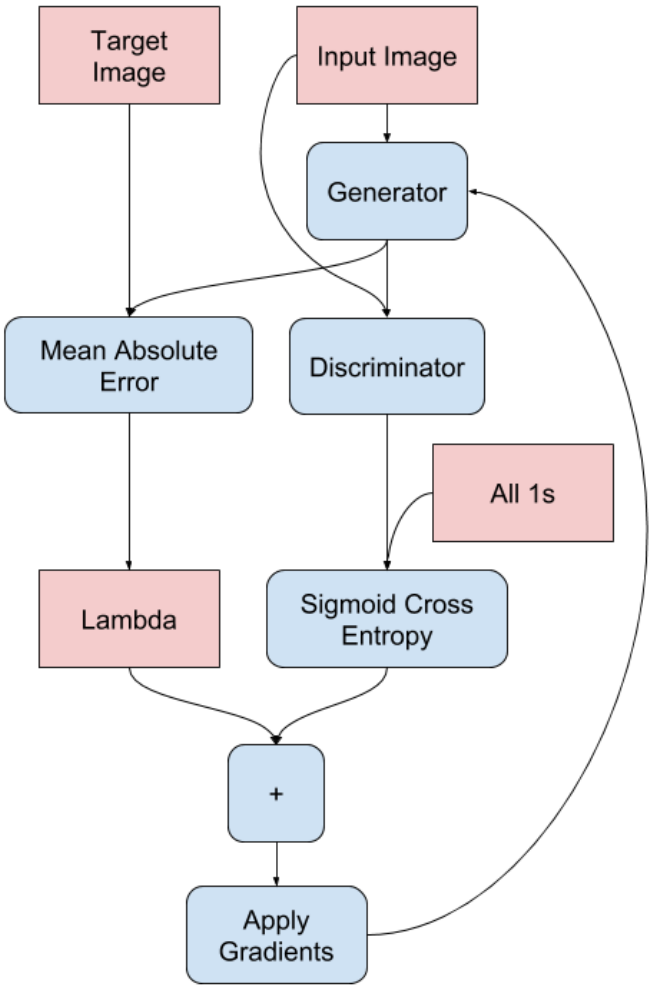

Figura 4.2: Pérdida del generador

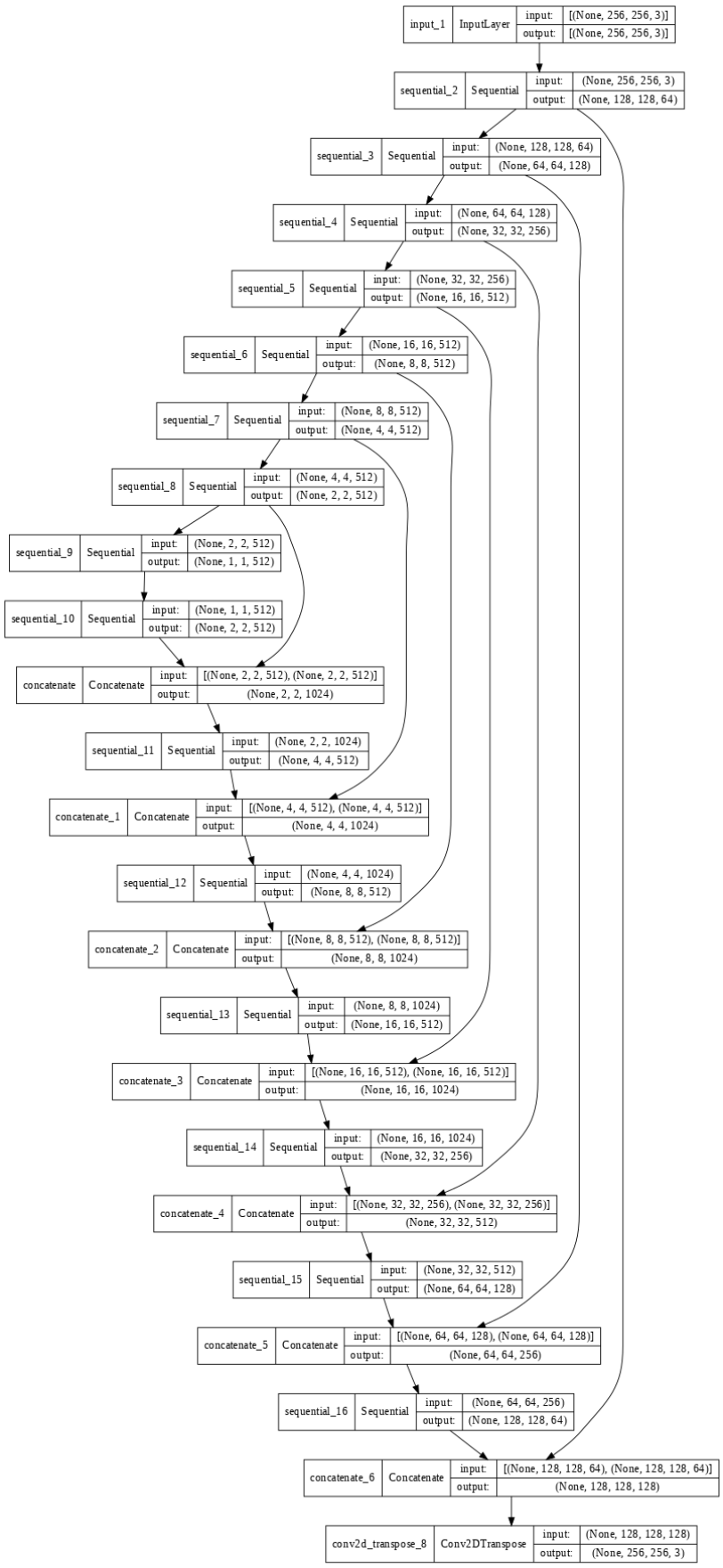

Figura 4.1: Arquitectura del generador

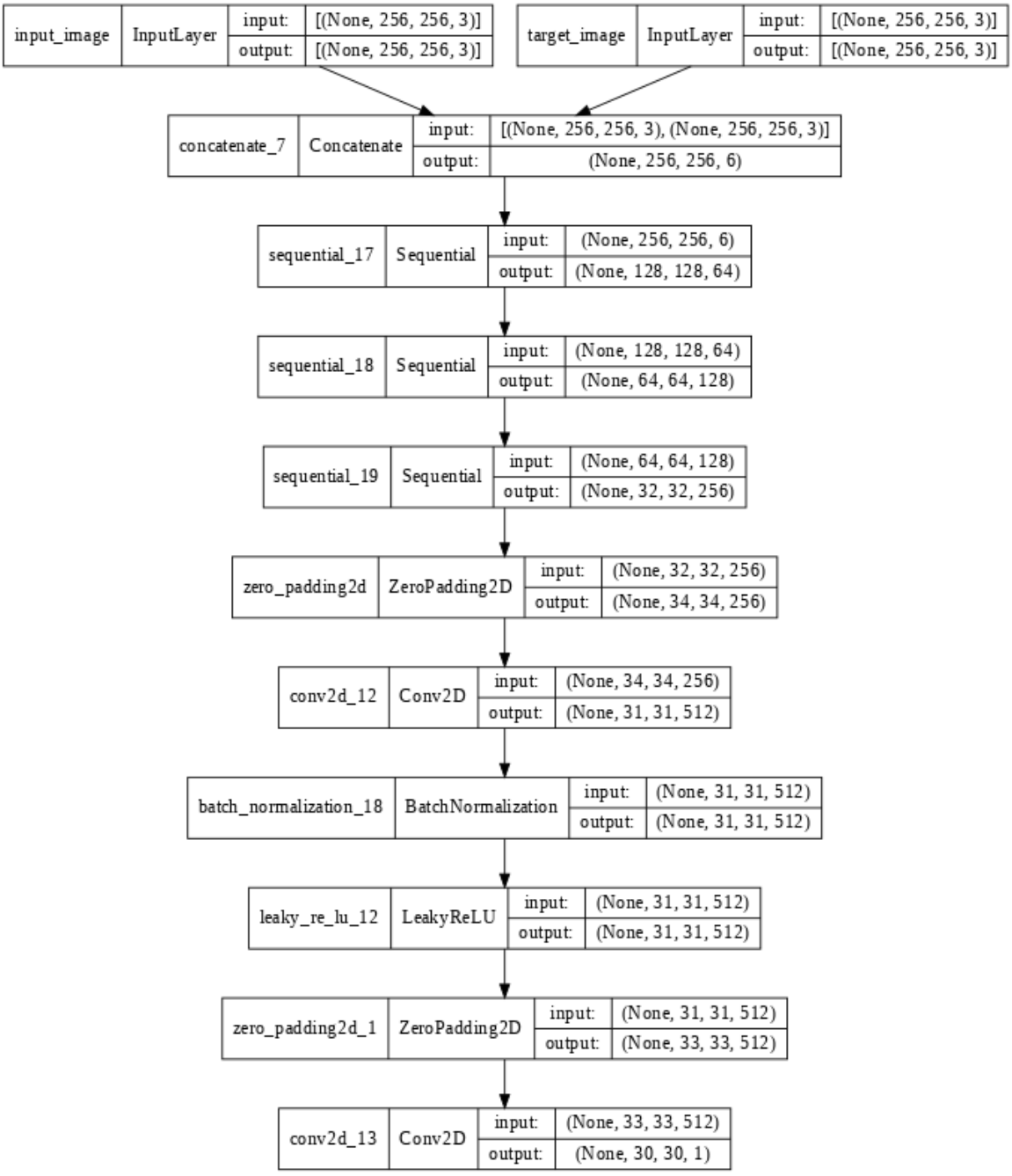

Figura 4.3: Arquitectura del discriminador

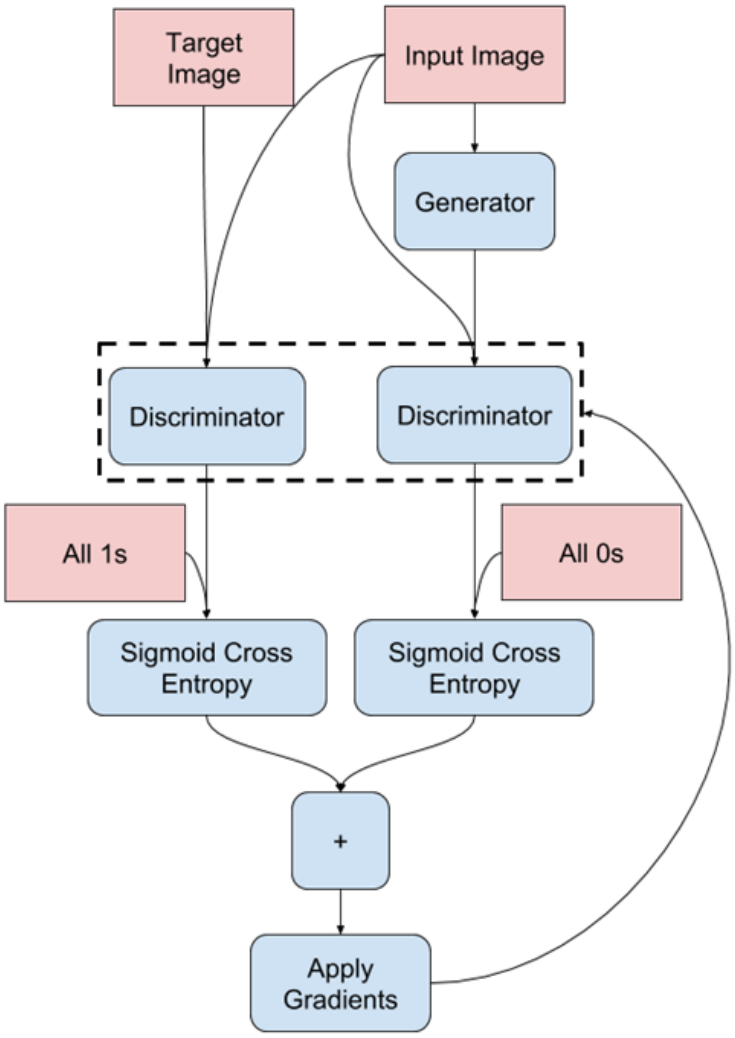

Figura 4.4: Pérdida del discriminador

4.2. Métricas de evaluación

Existen diversas técnicas de medición de la calidad de una imagen, las más comunes son las FR (Full Reference), donde se dispone de una imagen original, la cual no tiene ninguna alteración, y una imagen objetivo, la imagen previamente procesada, las métricas más populares son PSNR (Peak Signal to Noise Ratio) y SSIM (Structural Similarity Index Measure), que son ocupadas en una gran diversidad de problemas como se ve en [K.Silpa (2012)], aparte de estas métricas también se plantea usar EPI

(Edge Preservation Index), como su nombre lo indica está diseñada para analizar los cambios estructurales de una imagen y SNR (Signal to Noise Ratio) la cuál no depende de compara directamente dos imágenes.

4.2.1. Relación señal a ruido

La relación señal ruido o SNR (Signal to Noise Ratio) se define como la comparación del nivel de la señal deseado con el nivel del ruido de fondo. Cuanto mayor sea la relación, menos molesto será el ruido de fondo, se define por la Ecuación 4.1

$$SNR = \frac{\mu}{\sigma} \tag{4.1}$$

donde μ es el valor de la media de la imagen definida por la Ecuación **??**

$$\mu = \frac{1}{N}\sum_{i=0}^{N-1} x_i \tag{4.2}$$

y σ es la desviación estándar definida por la Ecuación 4.3

$$\sigma = \sqrt{\sigma^2} \tag{4.3}$$

$$\sigma^2 = \frac{1}{N-1}\sum_{i=0}^{N-1}(x_i - \mu)^2 \tag{4.4}$$

donde N es el total de píxeles y x_i es el píxel actual como se ve en [del Pilar Gómez Gil (2021)].

4.2.2. Proporción máxima de señal a ruido

Para la evaluación de la calidad se realiza habitualmente la comparación entre el error existente que hay entre las dos imágenes, siendo la métrica MSE (Mean Square

Error), y su variante más común PSNR (Peak Signal to Noise Ratio) descrita por la Ecuación 4.5, ampliamente utilizada en diversos proyectos como lo demuestran en [Javier Silvestre Blanes (2010)], dándonos cuantitativamente el efecto del ruido en una imagen.

$$PSNR = 20 * \log_{10}(\frac{MAX_f}{\sqrt{MSE}}) \tag{4.5}$$

donde MAX_f es el valor máximo que puede tomar un píxel de la imagen y MSE está definido como se ve en Ecuación 4.6.

$$MSE = \frac{1}{xy}\sum_{i=1}^{x}\sum_{j=1}^{y}(f(i,j) - g(i,j))^2 \tag{4.6}$$

En la cuál x es el ancho y y es el alto de la imagen, f y g son las imágenes a comparar. La implementación de está métrica se puede ver en el Apéndice A.1

4.2.3. Índice de similitud estructural

El índice de similitud estructural SSIM (Structural Similarity Index Measure) es una métrica utilizada para medir la similitud de dos imágenes, se basa en la percepción que considera la degradación de la imagen como un cambio percibido en la información estructural com se ve en [Javier Silvestre Blanes (2010)]. Usando la luminancia, el contraste y la estructura de las imágenes, definidas por la Ecuación 4.7, la Ecuación 4.8 y la Ecuación 4.9 respectivamente,

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{4.7}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{4.8}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x^2\sigma_y^2 + C_3} \tag{4.9}$$

si se toman estas tres ecuaciones y se multiplican entre si

$$SSIM(x,y) = (l(x,y))^{\alpha}(c(x,y))^{\beta}(s(x,y))^{\gamma} \tag{4.10}$$

donde $\alpha > 0, \beta > 0, \gamma > 0$ denotan la importancia relativa de cada una de las métricas. Asumimos, $\alpha = \beta = \gamma = 1$ y $C3 = C2/2$, para simplificar la expresión, podemos obtener el SSIM que está definido por la Ecuación 4.11.

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{4.11}$$

SSIM se calcula a partir de varias ventanas de la imagen, donde x y y son dos ventanas de tamaño N x N de la imagen, μ_x y μ_y son la media de los valores de los píxeles de las ventanas x y y, σ_x^2 y σ_y^2 son las varianzas de x y y, σ_{xy} es la covarianza de x y y, y C_1 esta definidas con la Ecuación 4.12 y C_2 esta definida con la Ecuación 4.13.

$$C_1 = (k_1 L)^2 \tag{4.12}$$

$$C_2 = (k_2 L)^2 \tag{4.13}$$

donde k_1 tiene el valor de 0.01 y k_2 el valor de 0.03 por defecto. L es el rango dinámico de los píxeles definido típicamente como $2^{numerodebitsporpixel} - 1$. Calculando el promedio de los valores SSIM se puede obtener el valor MSSIM. La implementación de está métrica se puede ver en el Apéndice A.2.

4.2.4. Índice de conservación de bordes

El índice de conservación de bordes EPI (Edge Preservation Index), parte de imágenes previamente procesadas con un filtro pasa altas, para localizar los bordes

de la imagen, en este caso se usa el filtro Laplaciano con un *kernel* de 3×3, el cual está definido por la Ecuación 4.14.

$$L(x, y) = \frac{\delta^2 I}{\delta x^2} + \frac{\delta^2 I}{\delta y^2} \tag{4.14}$$

donde $I(x, y)$ es la intensidad de un píxel en la imagen

$$\frac{\delta^2 I}{\delta x^2} = I(x+1, y) - 2I(x, y) + I(x-1, y) \tag{4.15}$$

$$\frac{\delta^2 I}{\delta y^2} = I(x, y+1) - 2I(x, y) + I(x, y-1) \tag{4.16}$$

Después se puede implementar la métrica que está definida por la Ecuación 4.17.

$$EPI = \frac{\sum(\Delta x - \bar{\Delta} x * \Delta y - \bar{\Delta} y)}{\sqrt{\sum(\Delta x - \bar{\Delta} x)^2 * \sum(\Delta y - \bar{\Delta} y)^2}} \tag{4.17}$$

donde Δx y Δy son las dos imágenes a comparar previamente filtradas y $\bar{\Delta} x$ y $\bar{\Delta} y$ son sus respectivas medias, como se puede ver en [Madan Lal (2016)] y en [Justin Joseph (2017)]. La implementación de está métrica se puede ver en el Apéndice A.3.

4.3. Herramientas utilizadas

Se usó python como lenguaje de programación para la implementación del proyecto, tiene un amplio catálogo de bibliotecas de Machine Learning y de ciencias en general, las bibliotecas con mayor importancia para realizar el proyecto fueron las siguientes:

La biblioteca de OpenCV, la cual está especializada en la visión computacional, su nombre viene de Open Source Computer Vision, la importancia del uso de esta biblioteca radica en el hecho que los datos que se usan para la entrada de la Red Neuronal Artificial son imágenes, las cuales se tuvieron que preprocesar para mejorar

el rendimiento de estas.

La biblioteca de TensorFlow, está diseñada para las tareas de Machine Learning, es desarrollada por Google, es la principal herramienta utilizada para el desarrollo, ya que en ella se diseña y entrena las RNA.

Capítulo 5

Desarrollo

5.1. Generación del conjunto de datos

Para este trabajo se usa una serie de imágenes de ultrasonido modo B compuestas principalmente por dos grandes conjuntos, cáncer de mama y cáncer de tiroides, ambos conjuntos cuentan con imágenes de tumores malignos y benignos, en menor cantidad se encuentran imágenes de distintos órganos humanos, teniendo una cantidad de 1,477 imágenes, estos conjuntos son públicos y se pueden consultar en [Al-Dhabyani W (2020)].

Para implementar el uso de esta arquitectura se necesita una imagen "original" y una "objetivo". La imagen original es la imagen de ultrasonido con ruido. La imagen objetivo, es la misma imagen pero con menos ruido. Esto puede representar un problema, pero en diferentes trabajos donde se ocupan las arquitecturas GAN lo solucionan modificando los datos originales, ya sea aplicando ruido o limpiando la imagen.

Para el proyecto se probó una combinación de estos dos métodos, se tomó el conjunto de datos con las imágenes sin alteración o "imágenes originales", agregando ruido *speckle* a esté y usando diversos filtro clásicos para limpiar las imágenes, al final se opto por usar las imágenes filtradas con el filtro clásico Fastnl, con el cual se obtuvo mejores resultados en las métricas de PSNR y SSIM.

Para la generación de ruido *speckle* se consultaron distintos trabajos, se optó por la implementación de acuerdo al *paper* [Singh y cols. (2017)] el cuál simula la adquisición

de la imagen de ultrasonido y cómo se genera el ruido *speckle*.

Hay dos técnicas de exploración por ultrasonido, exploración sectorial y exploración lineal. El escaneo de sector utiliza retrasos de transmisión en los elementos de la matriz para enfocar el haz y dirigirlo en una dirección particular. En el artículo, el tipo de sector de exploración del haz de ultrasonido se realiza muestreando una cuadrícula de píxeles. La dependencia de la resolución axial está en la longitud de onda y el número de repeticiones que forman los pulsos ultrasónicos. Entonces, el tamaño de una celda de resolución se basa en el hecho de que la resolución espacial de la imagen de ultrasonido es menor que la de la imagen. La pérdida de resolución axial debida a la longitud del pulso se simula muestreando la imagen.

El método de muestreo radial-uniforme genera m píxeles de muestreo equiespaciados a lo largo de la dirección radial como en el caso del muestreo radial-polar. Sin embargo, la falta de uniformidad en la distribución de puntos en el muestreo radial-polar se corrige manteniendo una distancia de arco constante entre los puntos a lo largo de cada arco. De este modo, obtenemos un patrón de muestreo que corresponde a la estructura geométrica del escaneo del sector mientras se conserva una distribución uniforme de puntos dentro de la región escaneada, como se ve en la Figura 5.1

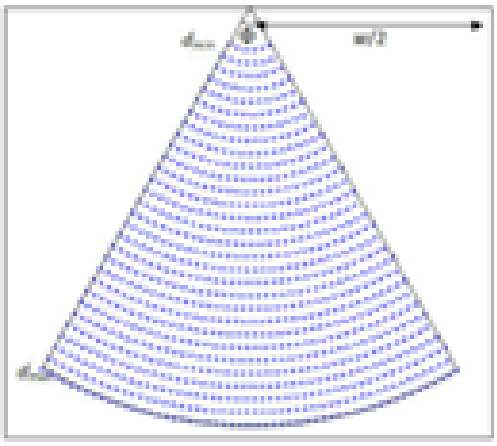

Figura 5.1: Muestreo radial uniforme, imagen tomada de Singh y cols. (2017).

Las operaciones de procesamiento de imágenes como la interpolación y la convolución que utilizan núcleos rectangulares requerirán que los píxeles estén dispuestos en una cuadrícula rectangular. Por lo tanto, una interpolación basada en el núcleo de

los puntos generados por el muestreo radial-polar requerirá una transformación de los puntos en una cuadrícula de $n \times m$, mostrada por Figura 5.2

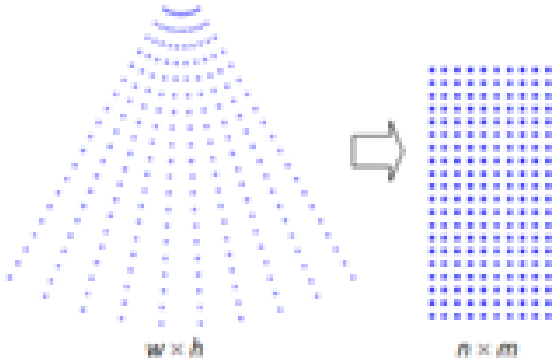

Figura 5.2: Transformación del muestreo en un rectángulo, imagen tomada de Singh y cols. (2017).

Para la simulación del ruido *speckle*, se adopta el método propuesto por Perreault y Auclair-Fortier. Su modelo se basa en una distribución compleja de fasores incoherentes (u, v) dada por una función gaussiana bidimensional g. La amplitud compleja de cada píxel se inicializa con la raíz cuadrada del valor de intensidad muestreado. El número de fasores incoherentes $M(x, y)$ en cada píxel (x, y) se establece como el valor de un número aleatorio bajo una distribución uniforme dentro de un rango preespecificado $[a, b]$. Los fasores incoherentes se generan y se agregan M veces a los componentes reales e imaginarios del valor complejo en cada píxel. El valor de la intensidad del ruido viene dada por la amplitud del número complejo. El pseudocódigo se puede ver en el Algoritmo 2.

Algorithm 2 Pseudocódigo para la generación de *speckle*

Input: μ, σ
Output: (u, v)
1: r_1, r_2: distribución uniforme aleatoria de números entre $[0, 1]$
2: **if** $r_1 < 0.00001$ **then**
3: $\quad r_1 = 0.00001$
4: $w_1 = \sqrt{-2\log(r_1)}\cos 2_2$
5: $w_2 = \sqrt{-2\log(r_1)}\sin 2_2$
6: $u = \mu + w_1\sigma$
7: $v = \mu + w_2\sigma$

La interpolación es la fase final de la generación del ruido *speckle*. Interpola las intensidades del nivel de gris ruidoso en los puntos muestreados para llenar el espacio vacío dejado por el paso de muestreo para obtener una imagen de sector completo como se ve en el Apéndice A.4.

Se procesaron las imágenes para generar distintos conjuntos de datos, las imágenes objetivo, que es el conjunto de datos original modificado con el filtrado de las imágenes dejándolas "limpias", y los conjuntos de datos de entrada, que es el conjunto de datos original, el conjunto de datos modificado con una adición de ruido *speckle*, en una cantidad baja y una cantidad alta. Como se muestra en la Figura 5.3

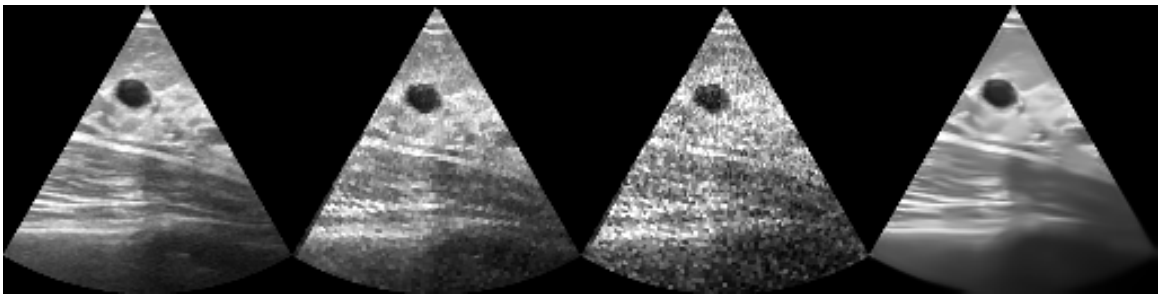

Figura 5.3: Ejemplo de los conjunto de datos obtenidos, de izquierda a derecha, imagen de entrada de la red, sin modificar, agregando poco ruido, agregando mucho ruido e imagen objetivo filtrada con el filtro Fastnl de la imagen original.

5.2. Implementación en Google Colab

El proyecto se desarrolló con Python usando la API de Tensorflow, en el entorno de desarrollo de Google Colab. Como el proyecto fue desarrollado en Google Colab, el cual borra la información guardada en él después de un tiempo, y se necesito realizar un manejo del conjunto de datos y el guardado de los pesos de las redes, se activó el uso de Google Drive en Google Colab.

Se preprocesó el conjunto de datos con una división del 80 % de entrenamiento y un 20 % de prueba, y se valido con las imágenes con ruido sin alterar, se reajustó el tamaño de la imagen y se normalizó, por último se aplicaron tuberías en Tensorflow para una mejor utilización de los datos. Se puede apreciar la implementación en el Apéndice A.5.

5.3. Replicación de la arquitectura

Para replicar la arquitecta del generador del *paper* se realizó una función para generar los bloques del codificador que está formado de [Convolución - Normalización de lote - Leaky ReLU], ver Apéndice A.6 y otra para generar los bloques del decodificador conformado por [Convolución transpuesta - Normalización de lote - Abandono (aplicado a los primeros 3 bloques) - ReLU], ver Apéndice A.7.

Teniendo el codificador y decodificador se implementó una estructura del tipo U-Net para el generador, ver Apéndice A.8, mientras que el discriminador es una estructura del tipo patchGAN, ver Apéndice A.9.

Se implementaron las funciones de pérdida para el generador y el discriminador, ver Apéndice A.10, las funciones de pérdida también son ocupadas en la red modificada, ya que estas no se modificaron.

Se implementó el optimizador para ambas redes, en la red modificada se usa la misma función de optimización pero con una tasa de aprendizaje distinta, ver Apéndice A.11.

5.4. Modificación de la arquitectura

Se consideró la arquitectura de la Figura 5.4 para la modificación, esta arquitectura es tomada del *paper* [Onur Karaoglu (2021)], la cual describe a groso modo la implementación de esta, por lo cual no es una replica exacta del trabajo, así que se tomo ciertas libertades para adaptarlo al conjunto de datos utilizado en el proyecto.

Se modificó la estructura de la arquitectura y algunos hiper-parámetros, como lo son la tasa de aprendizaje y los bloques que construyen al generador como su tamaño. Adicionalmente se implementó una serie de modificaciones al conjunto de datos con las imágenes de entrenamiento, girándolas, desacomodándolas y reflejándolas, algo que se propone en el *paper* [Phillip Isola (2018)], este proceso genera variabilidad en el entrenamiento de la red generando mejores resultados, donde se puede ver su implementación en el Apéndice A.12. Obteniendo como resultado imágenes de distintas

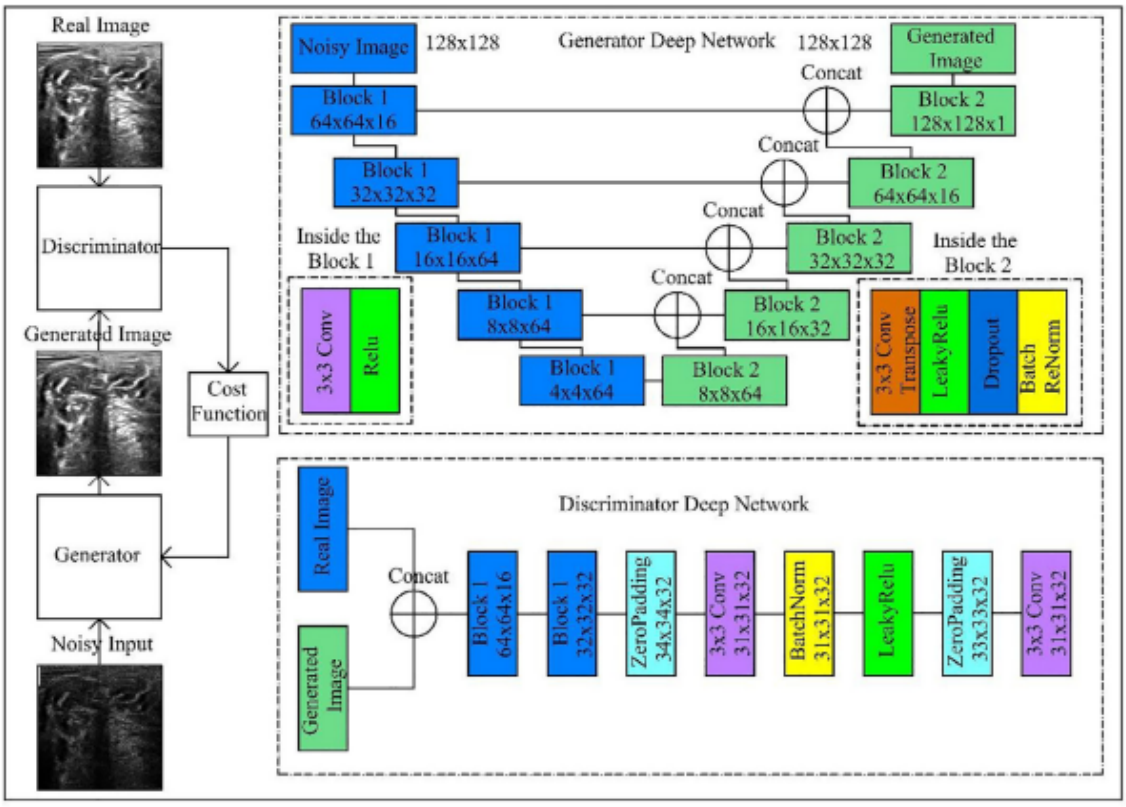

Fig. 4. Generative Adversarial Denoising Network schema.

Figura 5.4: Propuesta para la modificación, imagen tomada de Onur Karaoglu (2021).

formas como lo muestra la Figura 5.5:

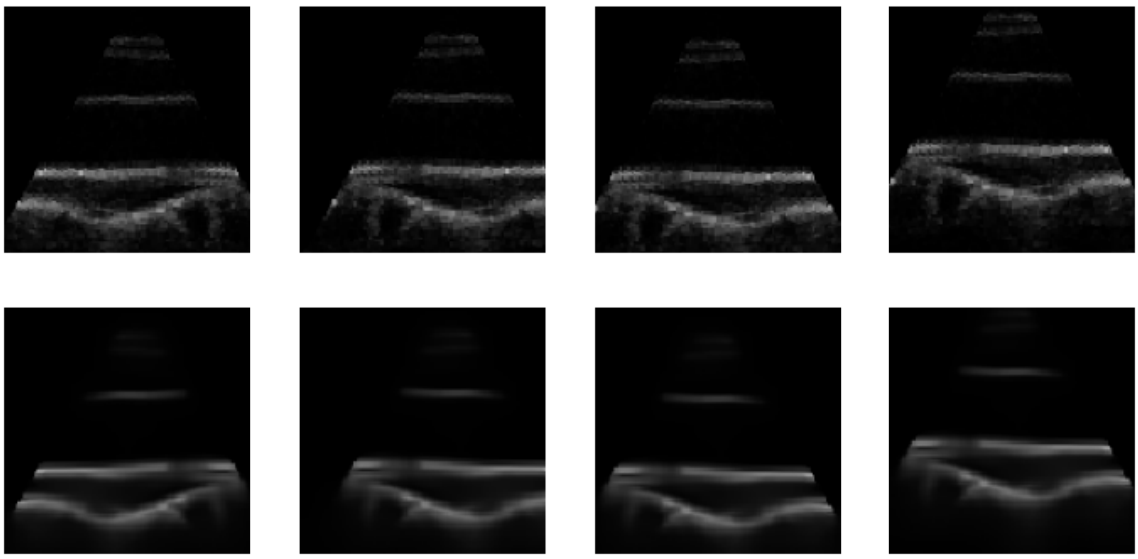

Figura 5.5: Acrecentamiento de datos

El proceso es el mismo que en la red que replica la arquitectura, la normalización y la redimensionalizacion del tamaño de las imágenes, en este caso las imágenes serán de 128×128, siendo esto un cambio respecto a la red pasada que eran de 256×256.

Para el generador se implementó el siguiente codificador visto en el Apéndice A.13 y decodificador visto en el Apéndice A.14. También se modificó la estructura del generador, ver Apéndice A.15 y del discriminador, ver Apéndice A.16. Las funciones de pérdida y los optimizadores se quedan igual que en la red que replica la arquitectura del *paper* [Phillip Isola (2018)], con la diferencia de la tasa de aprendizaje de 0.0001 en el optimizador ADAM.

El resultado de la arquitectura del generador y del discriminador se pueden ver en la Figura 5.6 y la Figura 5.7 respectivamente.

5.5. Implementación de un segundo conjunto de datos

Con el fin de buscar mejorar los resultados obtenidos, se propuso el uso de imágenes de radiografía, las cuales tienen una alta claridad en la estructura de los órganos internos.

En este caso se usaron imágenes de 5470 radiografías siendo en su mayoría imágenes de pulmones, el conjunto de datos se puede consultar en [Wang y cols. (2017)], se le agrego ruido *speckle* con el método antes mencionado del *paper* [Onur Karaoglu (2021)], una vez hecho esto se entrenó la red Pix2Pix y la red modificada, y se obtuvieron los valores de las métricas correspondientes como se ve en la Figura 5.8

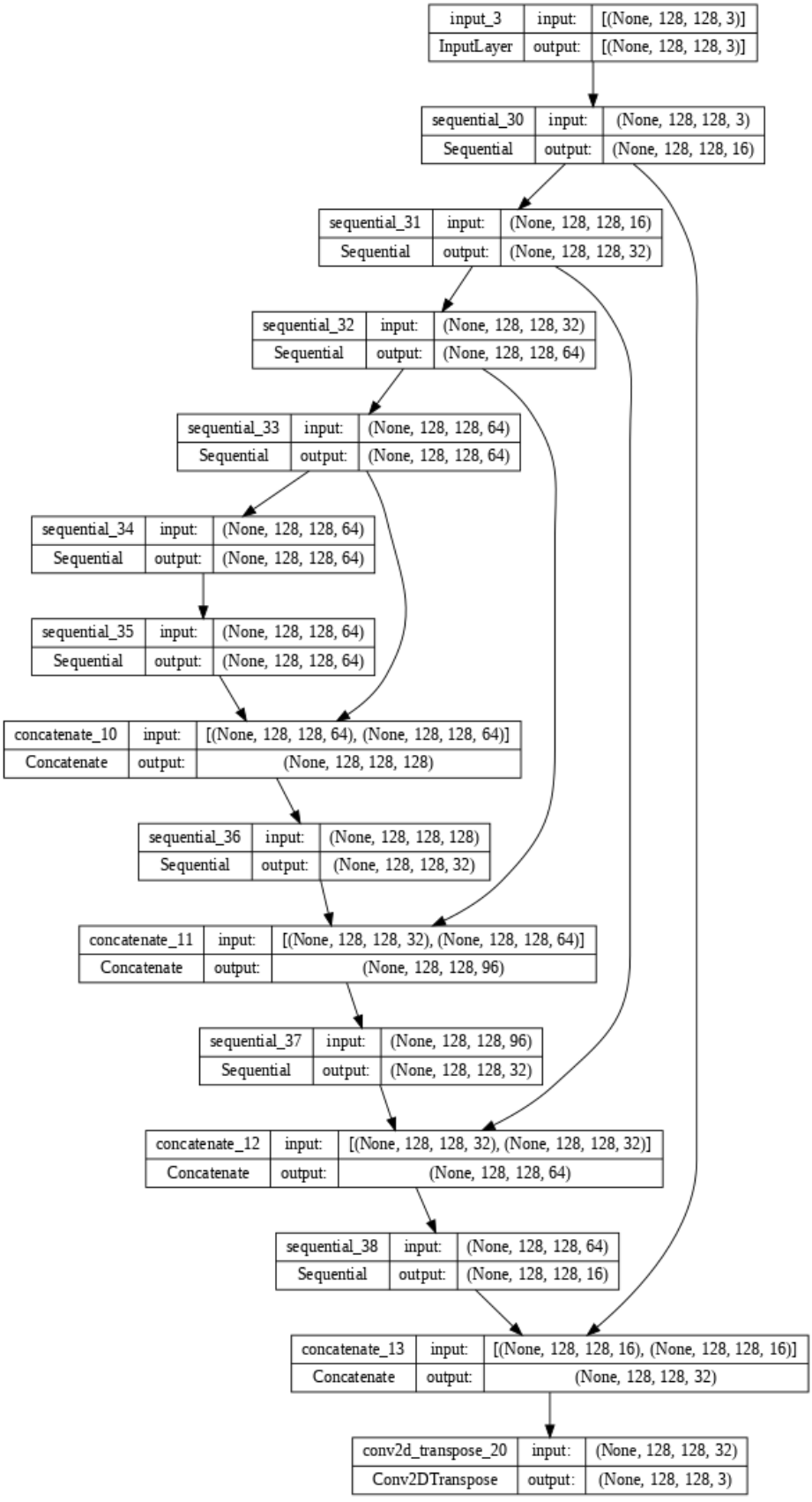

Figura 5.6: Arquitectura del generador modificado

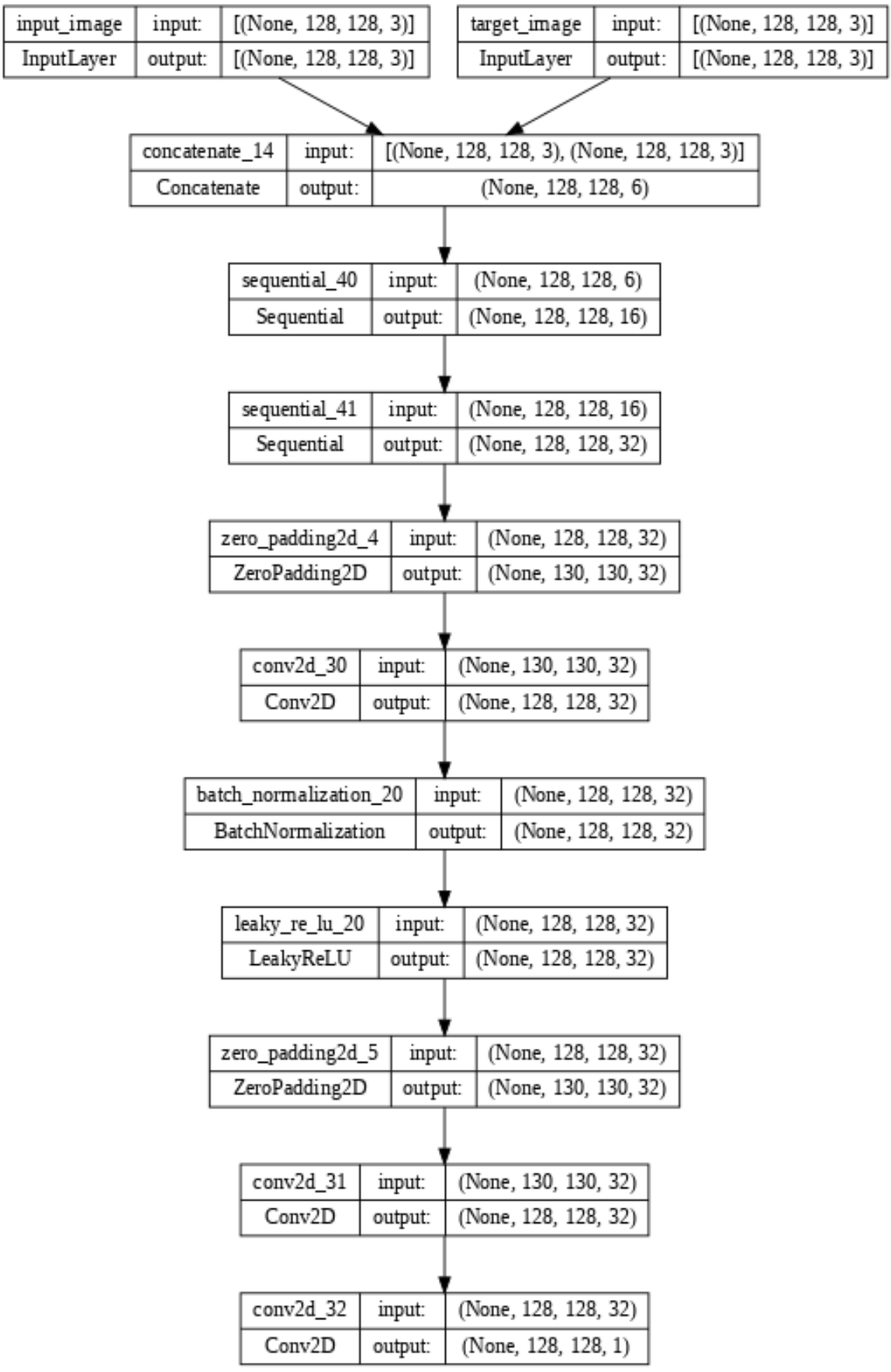

Figura 5.7: Arquitectura del discriminador modificado

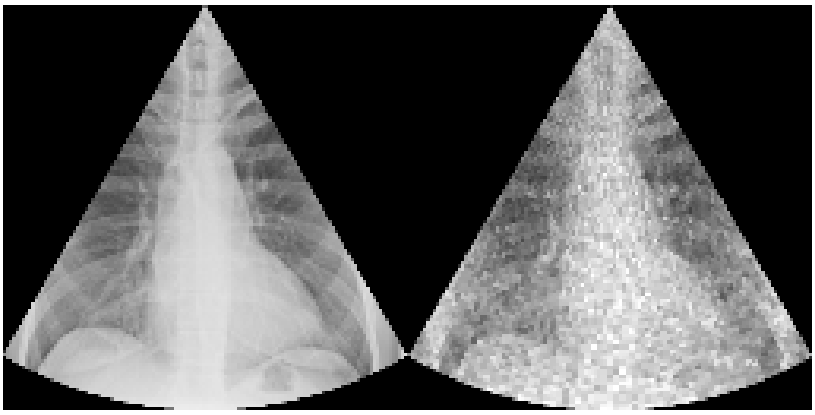

Figura 5.8: Conjunto de datos de imágenes de radiografía la imagen de la izquierda es la imagen sin alteraciones que se usará como "objetivo", a la imagen de la derecha se le agregó ruido *speckle* y se usa como la imagen de entrada de la red

Capítulo 6

Resultados y conclusiones

6.1. Resultados

A continuación se presentan los resultados obtenidos, se entrenaron en total 8 RNAs, lo cual significa que tendríamos 8 filtros uno por cada red entrenada, siendo 4 de ellas la replicación de la arquitectura del *paper* [Phillip Isola (2018)], con 4 distintos conjuntos de datos para el entrenamiento: imagen original de ultrasonido, imagen de ultrasonido con poco ruido agregado, imagen de ultrasonido con mucho ruido agregado e imagen de radiografía, y las otras 4 corresponden a la red modificada con los mismos conjunto de datos antes mencionados.

6.1.1. Replicación de la arquitectura

Se realizaron pruebas con diferentes conjuntos de datos de entrada para la RNA replicada, donde se utilizaron las imágenes de ultrasonido, sin alterar, con poco ruido agregado, con mucho ruido agregado e imágenes de radiografía, siendo las imágenes objetivo de ultrasonido la original filtrada con el filtro clásico Fastnl y para las imágenes de radiografía se usa la imagen original sin alterar. Los resultados de cada prueba se muestran en las Figuras 6.1, 6.2, 6.3 y 6.4, respectivamente. Se puede apreciar la comparación entre las dos imágenes de entrada (la imagen a limpiar y la imagen objetivo), y la imagen que genera la RNA.

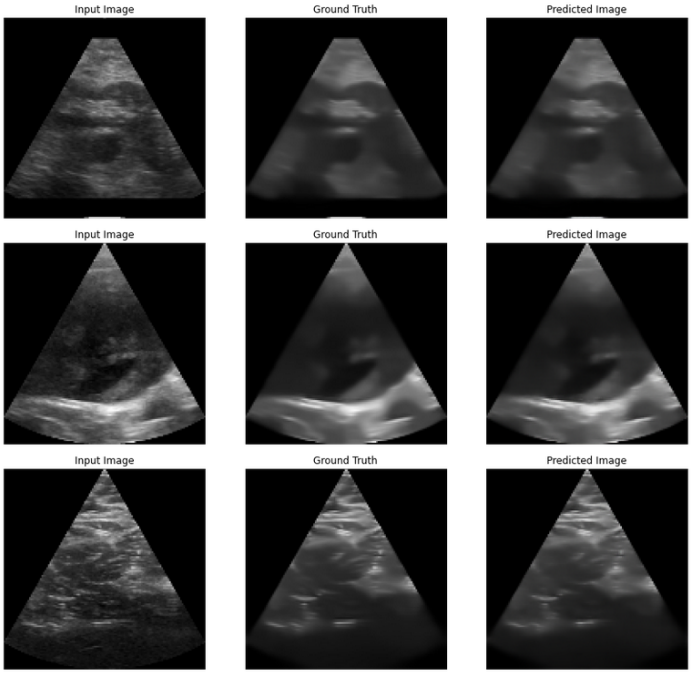

Figura 6.1: Resultados de la arquitectura de la red replicada, con imágenes de entrada sin alterar, de izquierda a derecha, imagen de entrada, imagen objetivo e imagen generada

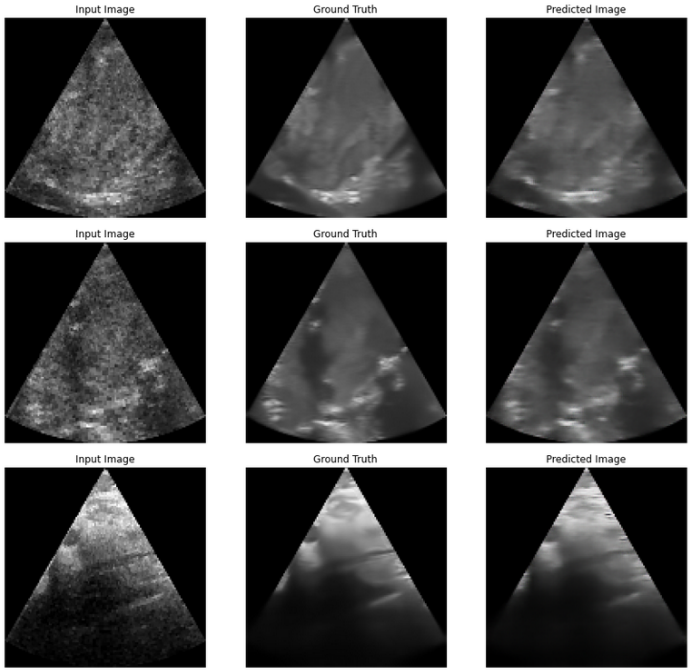

Figura 6.2: Resultados de la arquitectura de la red replicada, con imágenes de entrada a las que se les agregó poco ruido, de izquierda a derecha, imagen de entrada, imagen objetivo e imagen generada

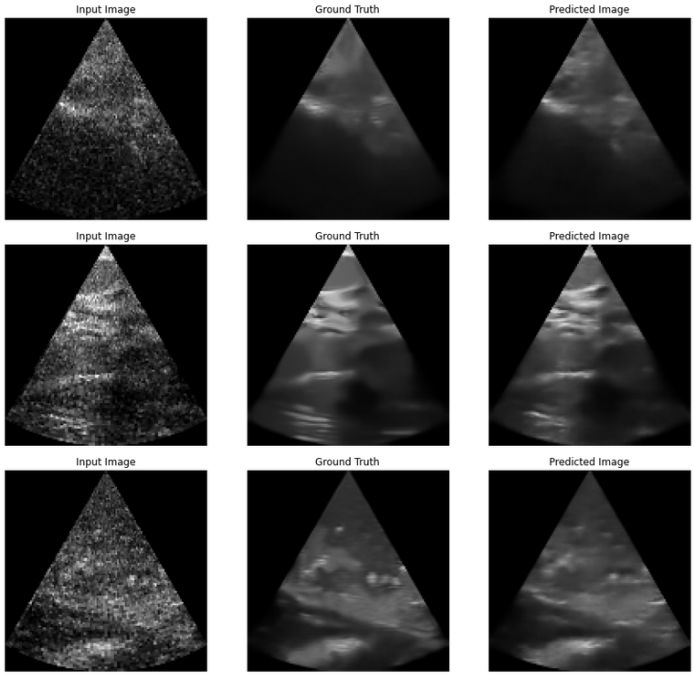

Figura 6.3: Resultados de la arquitectura de la red replicada, con imágenes de entrada a las que se les agregó mucho ruido, de izquierda a derecha, imagen de entrada, imagen objetivo e imagen generada

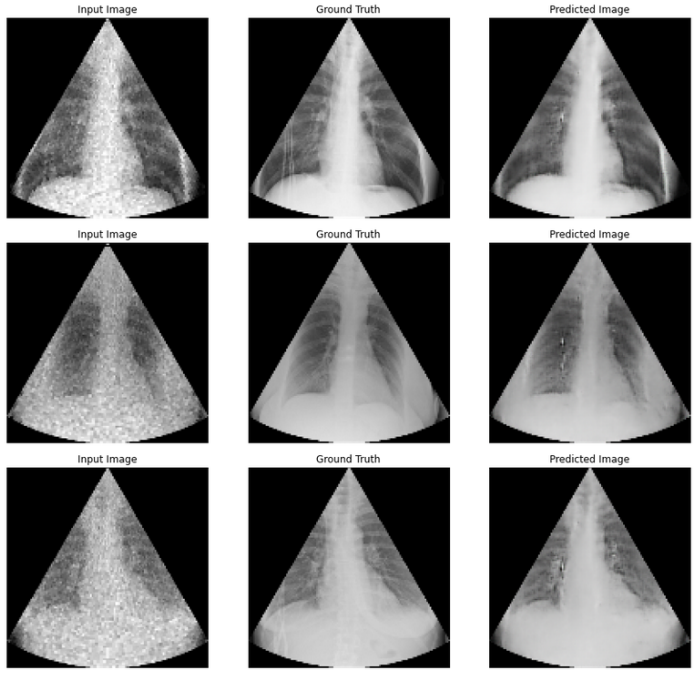

Figura 6.4: Resultados de la arquitectura de la red replicada, con imágenes de entrada de radiografía, de izquierda a derecha, imagen de entrada, imagen objetivo e imagen generada

6.1.2. Modificación de la arquitectura

Se realizaron pruebas con diferentes conjuntos de datos de entrada para la RNA modificada, donde se utilizaron las imágenes de ultrasonido, sin alterar, con poco ruido agregado, con mucho ruido agregado e imágenes de radiografía, siendo las imágenes objetivo de ultrasonido la original filtrada con el filtro clásico Fastnl y para las imágenes de radiografía se usa la imagen original sin alterar. Los resultados de cada prueba se muestran en las Figuras 6.5, 6.6, 6.7 y 6.8, respectivamente. Se puede apreciar la comparación entre las dos imágenes de entrada (la imagen a limpiar y la imagen objetivo), y la imagen que genera la RNA.

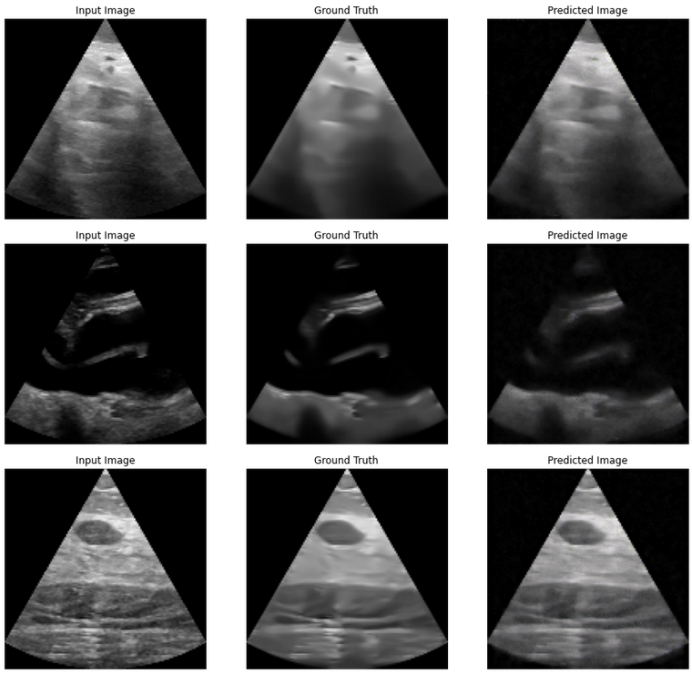

Figura 6.5: Resultados de la arquitectura de la red modificada, con imágenes de entrada sin alterar, de izquierda a derecha, imagen de entrada, imagen objetivo e imagen generada

6.1.3. Comparación entre los métodos de filtrado clásico y las RNAs propuestas

Se realizó una comparación donde se usaron filtros clásicos que se usan para la eliminación del ruido *speckle* y las métricas de SNR, PSNR, SSIM y EPI, con el fin de ver cuál es mejor para eliminar el ruido de la imagen y a la vez que conserve la

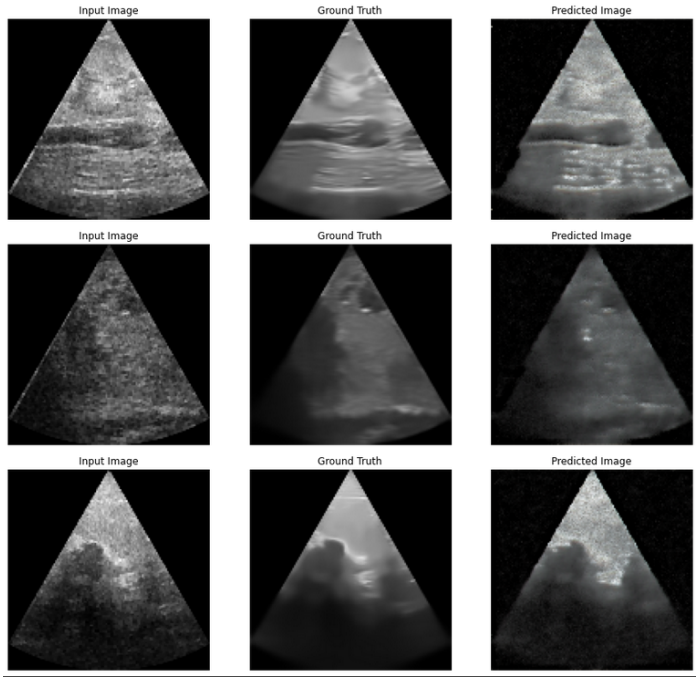

Figura 6.6: Resultados de la arquitectura de la red modificada, con imágenes de entrada a las que se les agregó poco ruido, de izquierda a derecha, imagen de entrada, imagen objetivo e imagen generada

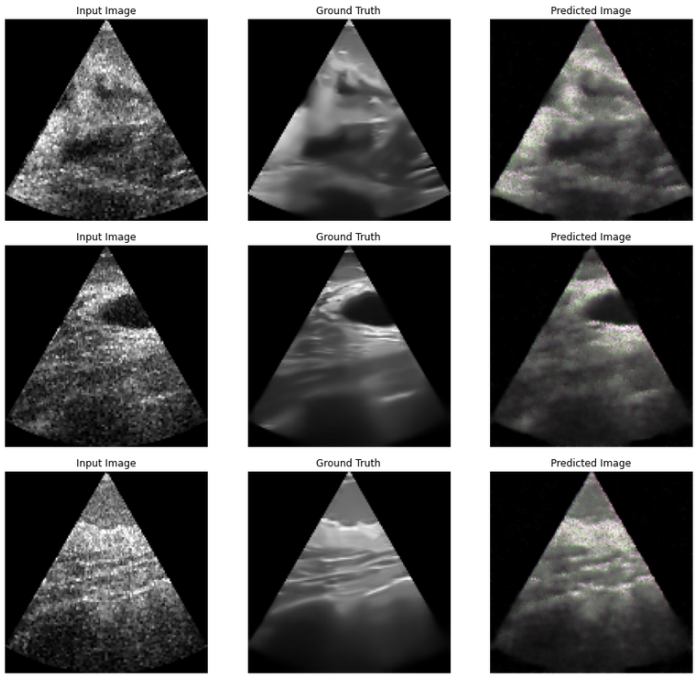

Figura 6.7: Resultados de la arquitectura de la red modificada, con imágenes de entrada a las que se les agregó mucho ruido, de izquierda a derecha, imagen de entrada, imagen objetivo e imagen generada

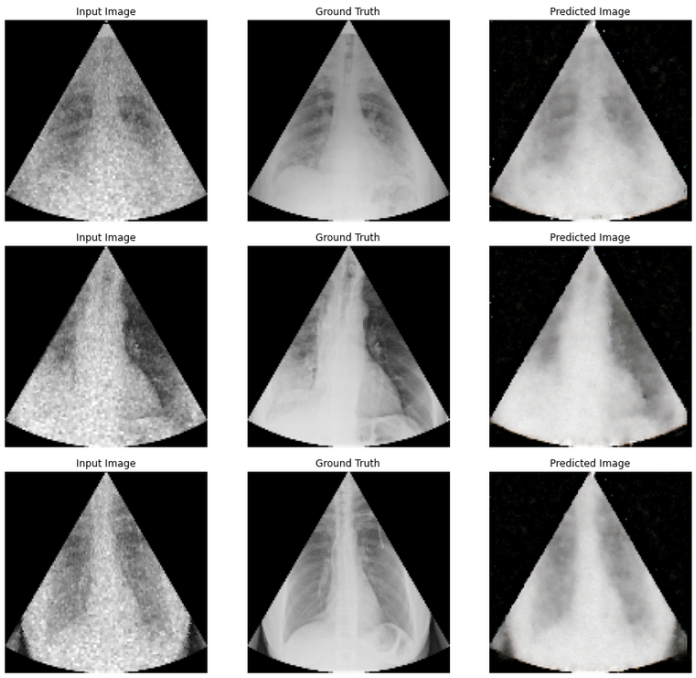

Figura 6.8: Resultados de la arquitectura de la red modificada, con imágenes de entrada de radiografía, de izquierda a derecha, imagen de entrada, imagen objetivo e imagen generada

estructura de la imagen, se puede apreciar una comparación de los resultados en la Figura 6.9.

Los valores obtenidos con las métricas se obtuvieron de un conjunto de 100 imágenes al azar del conjunto de datos de las imágenes de ultrasonido sin alterar, se filtraron con los filtros clásicos y filtros de RNAs propuestas en esté trabajo, comparando estos dos conjuntos se obtuvieron sus métricas de PSNR, SSIM y EPI, las cuales se promediaron para obtener los datos de la Tabla 6.1. Se puede ver en la Tabla 6.1 los valores obtenidos de las métricas PSNR qué nos indica el error causado por la diferencia entre dos imágenes, SSIM qué en esencia compara la diferencia estructural de dos imágenes y EPI que determina qué tan similares son dos imágenes dependiendo de sus bordes, siendo el valor de 100 % el que establece que las imágenes a comparar son exactamente las mismas. También se realizó la implementación de la métrica de SNR a cada una de las imágenes filtradas y a la imagen original, se puede ver los resultados en la Tabla 6.3.

Por la naturaleza del problema abordado en el proyecto, la eliminación del ruido *speckle* en imágenes de ultrasonido, no se cuenta con una referencia de imagen

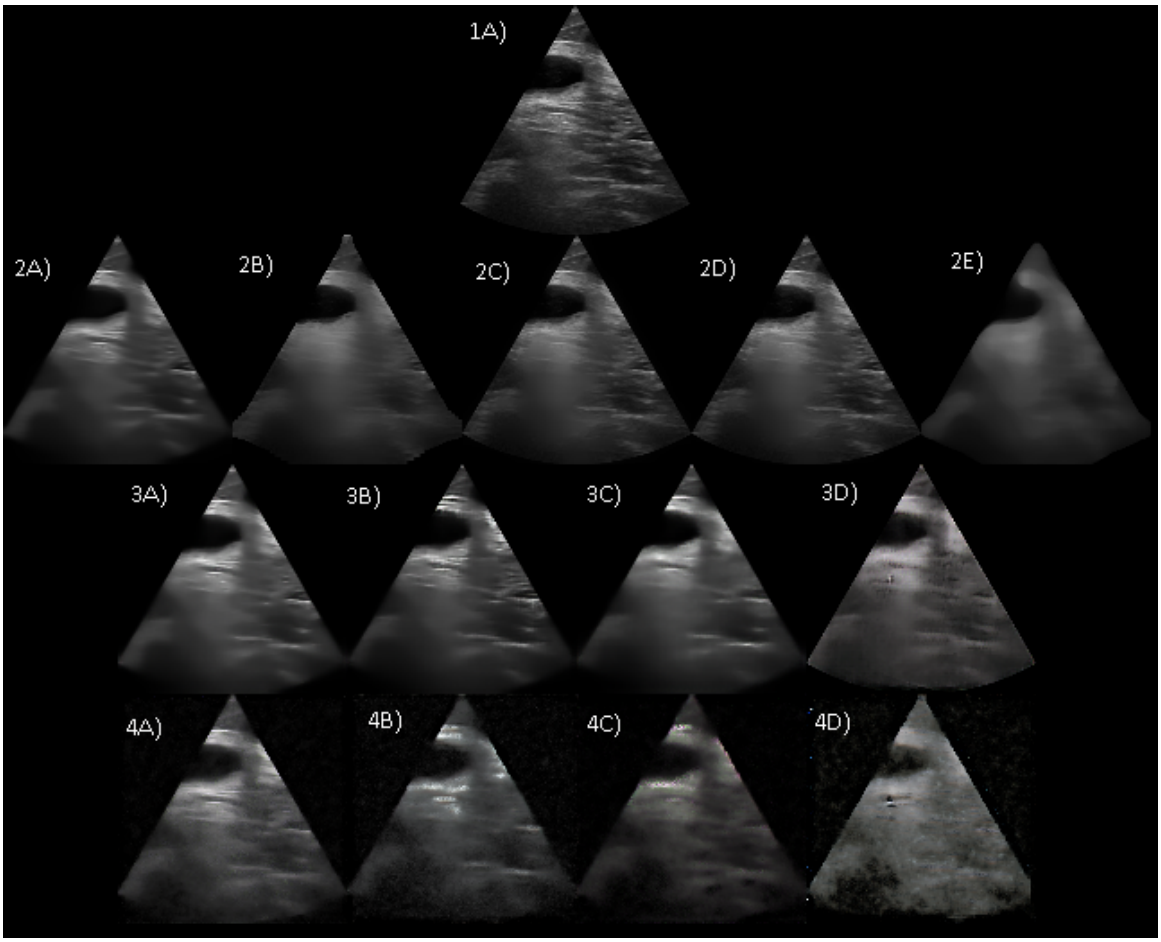

Figura 6.9: A) corresponde a una imagen de ultrasonido sin alteración, la fila 2 son los filtros clásicos, siendo A) Fastnl, B) Frost, C) Kuan, D) Lee y E) Median, la fila 3 son las RNAs con la arquitectura de Pix2Pix, la fila 4 son las RNAs con la arquitectura modificada, siendo estas dos filas, A) red entrenada con las imágenes de ultrasonido sin alteración, B) red entrenada con las imágenes de ultrasonido con poco ruido agregado, C) red entrenada con las imágenes de ultrasonido con mucho ruido agregado y D) red entrenada con las imágenes de radiografía

"ideal" con la cual comparar y a la cual desearíamos llegar, (obteniendo el 100 % en las métricas), así que la primera comparación se realizó con una imagen normal de ultrasonido que posee ruido *speckle* de la adquisición, por lo cual obtener el 100 % o números cercanos indicaría un bajo rendimiento en el filtrado, ya que estaría dejando la imagen muy parecida a la imagen sin filtrar.

Para tener más datos sobre el rendimiento de los filtros se optó por usar una nueva comparativa, esta vez se tomaron las imágenes de radiografía como referencia para la obtención de métricas, ya que se posee una imagen "ideal" que es la imagen original

FILTRO	PSNR	SSIM	EPI
Lee	35.37 %	87.28 %	87.84 %
Kuan	34.78 %	86.31 %	87.82 %
Frost	33.52 %	80.41 %	60.27 %
Median	32.86 %	69.89 %	19.18 %
Fastnl	34.17 %	85.47 %	82.68 %
RNA replicada con entrada sin modificar	32.10 %	86.41 %	84.06 %
RNA replicada con entrada de poco ruido	32.43 %	73.16 %	21.07 %
RNA replicada con entrada de mucho ruido	32.10 %	71.76 %	28.60 %
RNA replicada con entrada de imágenes de radiografía	31.75 %	72.23 %	38.89 %
RNA modificada con entrada sin modificar	30.56 %	57.88 %	74.59 %
RNA modificada con entrada de poco ruido	30.41 %	37.17 %	14.97 %
RNA modificada con entrada de mucho ruido	31.40 %	58.50 %	15.87 %
RNA modificada con entrada de imágenes de radiografía	29.71 %	36.13 %	31.96 %

Tabla 6.1: Comparación entre diferentes filtros y las redes propuestas, para imágenes de ultrasonido

sin alterar y una imagen con ruido *speckle* a la cual filtrar, se usó la misma dinámica de elegir 100 imágenes al azar, teniendo como resultado los valores de la Tabla 6.2.

FILTRO	PSNR	SSIM	EPI
Lee	32.84 %	84.68 %	68.28 %
Kuan	32.61 %	83.78 %	68.28 %
Frost	32.74 %	83.94 %	33.85 %
Median	33.04 %	81.59 %	18.26 %
Fastnl	33.96 %	86.98 %	77.21 %
RNA replicada con entrada sin modificar	31.20 %	82.64 %	80.08 %
RNA replicada con entrada de poco ruido	32.72 %	89.13 %	86.14 %
RNA replicada con entrada de mucho ruido	32.19 %	87.80 %	85.13 %
RNA replicada con entrada de imágenes de radiografía	31.04 %	92.18 %	91.56 %
RNA modificada con entrada sin modificar	30.81 %	57.01 %	66.16 %
RNA modificada con entrada de poco ruido	31.31 %	56.17 %	31.49 %
RNA modificada con entrada de mucho ruido	31.09 %	62.11 %	22.10 %
RNA modificada con entrada de imágenes de radiografía	30.09 %	63.29 %	73.66 %

Tabla 6.2: Comparación entre diferentes filtros y las redes propuestas, para imágenes de radiografía

FILTRO	SNR	μ	σ
Imagen original	0.7592	34.5629	45.3664
Lee	0.7885	34.1912	43.2634
Kuan	0.7966	34.2197	42.8691
Frost	0.8034	33.7684	41.9383
Median	0.7974	32.2663	40.3750
Fastnl	0.7763	34.1905	44.0274
RNA replicada con entrada sin modificar	0.7764	40.7671	52.1698
RNA replicada con entrada de poco ruido	0.7494	35.8408	47.4515
RNA replicada con entrada de mucho ruido	0.7530	37.3675	49.2433
RNA replicada con entrada de imágenes de radiografía	0.8643	44.8504	51.7995
RNA modificada con entrada sin modificar	0.8804	39.5449	45.5020
RNA modificada con entrada de poco ruido	0.7881	33.1840	42.3749
RNA modificada con entrada de mucho ruido	0.7954	39.7256	50.0299
RNA modificada con entrada de imágenes de radiografía	0.9349	50.7673	54.1963

Tabla 6.3: Comparación entre la imagen original, filtros clásicos y las RNAs propuestas, para la métrica de SNR y sus componentes

6.2. Análisis de los resultados

Tomando los datos de la Tabla 6.1 se puede intuir que los filtros con un porcentaje menor son los que tendrían un mejor funcionamiento, ya que se alejan de la imagen con ruido, esto se podría comprobar al comparar la imagen original contra el mejor y peor filtro obtenido, tomando como referencia el promedio de sus métricas, Tabla 6.4.

Al comparar las dos imágenes filtradas, como se ve en la Tabla 6.5, se puede deducir que efectivamente existe más relación entre la imagen original con el ruido *speckle* y la imagen filtrada por el filtro Lee, que entre la imagen original y la imagen filtrada por RNA, se puede apreciar de forma visual en la Figura 6.10. El razonamiento anterior no nos indica que este sea el mejor filtro, alejarse de la imagen original no necesariamente es lo mejor, ya que una imagen totalmente diferente puede tener los mismos resultados en las métricas, por eso es necesario observar la Tabla 6.2 donde se puede ver de una manera más objetiva el rendimiento de cada filtro, centrándonos en el valor EPI que es el que nos interesa para el filtrado de la imagen. Considerando ambos experimentos y los resultados se puede ver que la RNA con mayor rendimiento

FILTRO	promedio de las métricas
Lee	70.16
Kuan	69.63
Frost	58.06
Median	40.64
Fastnl	67.44
RNA replicada con entrada sin modificar	67.52
RNA replicada con entrada de poco ruido	42.36
RNA replicada con entrada de mucho ruido	44.15
RNA replicada con entrada de imágenes de radiografía	47.62
RNA modificada con entrada sin modificar	54.37
RNA modificada con entrada de poco ruido	27.51
RNA modificada con entrada de mucho ruido	35.25
RNA modificada con entrada de imágenes de radiografía	32.60

Tabla 6.4: Promedio de las métricas de la comparación para imágenes de ultrasonido.

sería la replicada que tuvo un entrenamiento con imágenes de radiografía que se les agregó poco ruido para el entrenamiento, se pueden ver sus resultados en la Figura 6.11.

Sobre los resultados de la métrica SNR en la Tabla 6.3 se puede ver que la relación señal a ruido mayor es la RNA modificada entrenada con imágenes de radiografía pero su media dista mucho de la imagen original, observando las métricas de la media y desviación estándar que no varíen en gran medida respecto a la imagen original tenemos a la RNA modificada con entrada sin modificar.

Comparación	PSNR	SSIM	EPI
Imagen Original vs Filtro Clásico Lee	35.37 %	87.28 %	87.84 %
Imagen Original vs Filtro RNA modificada poco ruido	30.41 %	37.17 %	14.97 %
Filtro Clásico Lee vs Filtro RNA modificada poco ruido	30.71 %	44.18 %	22.00 %

Tabla 6.5: Comparación entre filtro clásico y filtro RNA, como el resultado de la comparación de ambos filtros

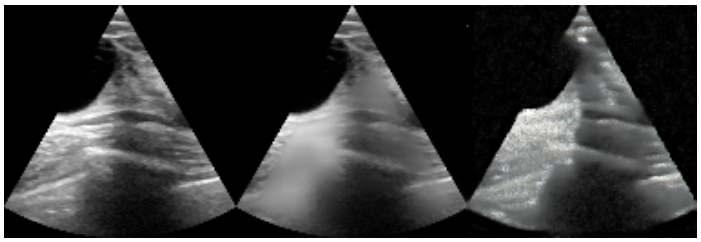

Figura 6.10: De izquierda a derecha, imagen original de ultrasonido sin alterar, imagen filtrada con Lee, imagen filtrada con la RNA modificada y entrenada con imágenes con poco ruido agregado

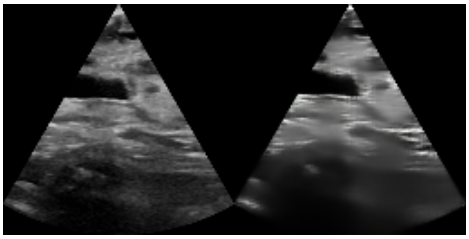

Figura 6.11: A la izquierda imagen original de ultrasonido sin alterar,a la derecha, imagen filtrada con la RNA replicada que se entreno con imágenes de radiografía a las cuales se les agregó poco ruido para el entrenamiento

6.3. Conclusiones y trabajo a futuro

El objetivo de este trabajo fue determinar la eficiencia de las RNAs del tipo GAN para la tarea de eliminación de ruido *spckle* en imágenes de ultrasonido, dándole prioridad a la conservación estructural de la imagen.

Se han utilizado varios conjuntos de datos de entrenamiento y se ha empleado una red GAN previamente propuesta en el *paper* [Phillip Isola (2018)] que ha demostrado excelentes resultados para diversas tareas de procesamiento de imágenes, así como lo son la síntesis de imágenes a partir de mapas de segmentos, la generación de imágenes a color a partir de imágenes en blanco y negro, la tarea de completar imágenes con espacios vacíos, entre muchas otras. También se realizo una segunda implementación basada en la modificación de dicha arquitectura.

El uso de las GANs en comparación con los métodos clásicos de filtrado ha demostrado ser mejor en la tarea de filtrado de imágenes de ultrasonido. Aunque el resultado obtenido aún no es excepcional, existe un margen suficiente de mejora para considerar la adopción de estos nuevos métodos de filtrado. Esto sugiere que las GANs

son una herramienta prometedora en la tarea de procesamiento de imágenes y que todavía hay un gran potencial para mejorar el rendimiento de estos nuevos filtros. Por lo tanto, se espera que en el futuro se utilicen cada vez más estas nuevas técnicas de filtrado.

Después de llevar a cabo este trabajo de tesis, al igual que en cualquier otra investigación, hay varias líneas de investigación que aún están abiertas y que se pueden explorar en el futuro. Durante el desarrollo, surgieron algunas líneas futuras relacionadas directamente con el trabajo de tesis y otras más generales que podrían ser de interés para futuros investigadores. También se proponen algunos desarrollos específicos para respaldar y mejorar el modelo y la metodología presentada. Dentro de los posibles trabajos futuros, se identifican algunos en particular que se consideran destacados:

- La adquisición de un conjunto de datos, donde se tengan varias tomas de un mismo punto espacial, las cuales solo variarían en el ruido adquirido, teniendo así un lote de imágenes que les corresponde una única imagen objetivo.
- Construir un conjunto de datos de entrenamiento variado con distintas formas de implementación de ruido *speckle*, y distintos filtrados, lo cual ayudaría a generalizar la tarea de filtrado que realiza la red.
- Implementar un módulo al final de *supersampling* [Xintao Wang (2017)], que puede ayudar a mejorar en general la calidad de la imagen.
- A lo largo de la realización de este proyecto, se han desarrollado nuevos modelos de aprendizaje de máquina, con un enfoque en la generación de imágenes, se puede explorar la factibilidad de la implementación de alguno de ellos.
- La generación de una métrica del tipo RR (Reduced-Reference), que se enfoque en la conservación estructural de las imágenes, partiendo del hecho de que no se puede adquirir imágenes limpias de ultrasonido.
- Otra métrica que se podría usar es tomar un sistema de segmentación que se usara para segmentar las imágenes filtradas, una vez hecho esto se puede ir

comparando las segmentaciones realizadas sobre los distintos filtros aplicados para ir comparando.

Referencias

admin. (s.f.). *Entrenamiento de redes neuronales b) descenso de gradiente.* Descargado de https://logongas.es/doku.php?id=clase:iabd:pia:2eval:tema07.backpropagation_descenso_gradiente

Al-Dhabyani W, K. H. F. A., Gomaa M. (2020). *Dataset of breast ultrasound images.* Descargado de https://www.kaggle.com/datasets/aryashah2k/breast-ultrasound-images-dataset

Antoni Buades, J.-M. M., Bartomeu Coll. (2015, June). Non-local means denoising. *Image Processing On Line*, 5. Descargado de https://www.ipol.im/pub/art/2011/bcm_nlm/article.pdf

Bonifacio Martín del Brío, A. S. M. (2007). *Redes neuronales y sistemas borrosos.* Alfaomega.

Carovac A, J. D., Smajlovic F. (2011, September). *Application of ultrasound in medicine.* Descargado de https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3564184

del Pilar Gómez Gil, M. (2021). *Procesamiento digital de señales semana 1. introducción.* Descargado de https://ccc.inaoep.mx/~pgomez/cursos/pds/slides/S1B-Est.pdf

Dietrichson, F., Smistad, E., Ostvik, A., y Lovstakken, L. (2018). Ultrasound speckle reduction using generative adversial networks. , 1-4. Descargado de https://ieeexplore.ieee.org/document/8579764 doi: 10.1109/ULTSYM.2018.8579764

Erick Cuevas, M. P., Daniel Zaldivar. (2010). *Procesamiento digital de imágenes con matlab y simulink.* Alfaomega.

Esteves, T. (2020). *Extraindo representações com autoencoders convolucionais.* Descargado de https://estevestoni.medium.com/extraindo-representa%C3%A7%C3%B5es-com-autoencoders-convolucionais-405ab73afa05

Guo, H., Nguyen, H., Vu, D.-A., y Bui, X.-N. (2021). Forecasting mining capital cost for open-pit mining projects based on artificial neural network approach. *Resources Policy*, *74*, 101474. Descargado de https://www.sciencedirect.com/science/article/pii/S0301420718306901 doi: https://doi.org/10.1016/j.resourpol.2019.101474

Javier Silvestre Blanes, J. L. G. (2010). *Técnicas de evaluación de la calidad de la imagen. tendencias y métricas basadas en bordes.* Descargado de https://dialnet.unirioja.es/servlet/articulo?codigo=6435028 (Instituto Tecnológico de Informática (ITI), Universidad Politécnica de Valencia (UPV), Universidad Politécnica de Cataluña (UPC))

Justin Joseph, R. P. V. S., Sivaraman Jayaraman. (2017, January). *An edge preservation index for evaluating nonlinear spatial restoration in mr images.* Descargado de https://www.researchgate.net/publication/301558791_An_Edge_Preservation_Index_for_Evaluating_Nonlinear_Spatial_Restoration_in_MR_Images (Current Medical Imaging Reviews · January 2017)

KeepCoding. (2022). *¿qué es una función de activación en deep learning?* Descargado de https://keepcoding.io/blog/funcion-de-activacion-en-deep-learning/

K.Silpa, D. M. (2012, June). Comparison of image quality metrics. *International Journal of Engineering Research Technology (IJERT)*, *1*, 5. Descargado de https://www.ijert.org/research/comparison-of-image-quality-metrics-IJERTV1IS4105.pdf

Lei Zhu, M. S. B. P.-A. H., Chi-Wing Fu. (s.f.). A non-local low-rank framework for ultrasound speckle reduction. *IEEE*. Descargado de https://openaccess.thecvf.com/content_cvpr_2017/papers/Zhu_A_Non-Local_Low-Rank_CVPR_2017_paper.pdf

Madan Lal, S. G., Lakhwinder Kau. (2016). *Speckle reduction with edge preservation in b- scan breast ultrasound image.* Descargado de https://www.mecs-press.org/ijigsp/ijigsp-v8-n9/IJIGSP-V8-N9-8.pdf (Modern Education and Computer Science PRESS)

MedImaging. (2019). *Existe una inclinación creciente hacia los equipos manuales de ultrasonido para los poc.* Descargado de https://mobile.medimaging.es/industria/articles/294777177/existe-una-inclinacion-creciente-hacia-los-equipos-manuales-de-ultrasonido-para-los-poc.html

Onur Karaoglu, I. U., Hasan Sſakir Bilge. (2021, July). Removal of speckle noises from ultrasound images using five different deep learning networks. *ELSEVIER*, 12. Descargado de https://www.sciencedirect.com/science/article/pii/S2215098621001427

Phillip Isola, T. Z. A. A. E., Jun-Yan Zhu. (2018, November). Image-to-image translation with conditional adversa- rial networks. Descargado de https://arxiv.org/abs/1611.07004 (Berkeley AI Research (BAIR) Laboratory, UC Berkeley)

Ponce, J. (2021). *Conoce qué son las funciones de activación y cómo puedes crear tu función de activación usando python, r y tensorflow.* Descargado de https://jahazielponce.com/funciones-de-activacion-y-como-puedes-crear-la-tuya-usando-python-r-y-tensorflow/

P.S. Hiremath, P. T. A., y Badiger, S. (2013). *Speckle noise reduction in medical ultrasound images.* IntechOpen. Descargado de https://www.intechopen.com/chapters/45101 doi: 10.5772/56519

P.S. Hiremath, S. B., Prema T. Akkasaligar. (2013). *Advancements and breakthroughs in ultrasound imaging.* IntechOpen. Descargado de https://www.intechopen.com/chapters/45101

Singh, P., Mukundan, R., y de Ryke, R. (2017). Synthetic models of ultrasound image formation for speckle noise simulation and analysis. , 278-284. Descargado de https://ieeexplore.ieee.org/document/7967056 doi: 10.1109/ICSIGSYS.2017.7967056

TensorFlow. (s.f.). *pix2pix: Image-to-image translation with a conditional gan.* Descargado de https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/generative/pix2pix.ipynb#scrollTo=_xnMOsbqHz61

Wang, X., Peng, Y., Lu, L., Lu, Z., Bagheri, M., y Summers, R. (2017). Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. En *2017 ieee conference on computer vision and pattern recognition(cvpr)* (pp. 3462–3471). Descargado de https://academictorrents.com/details/557481faacd824c83fbf57dcf7b6da9383b3235a

Xintao Wang, C. D. Y. S., Liangbin Xie. (2017, August). *Real-esrgan: Training real-world blind super-resolution with pure synthetic data.* Descargado de https://arxiv.org/pdf/2107.10833.pdf (Applied Research Center (ARC), Tencent PCG, Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, University of Chinese Academy of Sciences, Shanghai AI Laboratory)

Apéndice A

Código

```
def PSNR(original, target):
    mse = np.mean((original - target) ** 2)
    if(mse == 0):
        return 100
    max_pixel = 255.0
    psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr
```

Listing A.1: Implementación de la proporción máxima de señal a ruido

```
def mssim(img_1, img_2):
    c1 = (0.01*255)**2
    c2 = (0.03*255)**2
    win = gaussian(11, 1.5)

    mu1 = filters.correlate(img_1, win)
    mu1_sq = mu1*mu1;
    s1sq =filters.correlate(img_1*img_1, win)-mu1_sq

    mu2 = filters.correlate(img_2, win)
    mu2_sq = mu2*mu2;
    mu1_mu2 = mu1*mu2;

    s2sq = filters.correlate(img_2*img_2, win)-mu2_sq
    s12 = filters.correlate(img_1*img_2, win)-mu1_mu2
```

```
16
17      ssims = ((2*mu1_mu2 + c1)*(2*s12 + c2))/((mu1_sq + mu2_sq + c1)
        *(s1sq + s2sq + c2))
18
19      return ssims.mean()
```

Listing A.2: Implementación de la medida del índice de similitud estructural

```
1  def EPI(image_i, image_f):
2    ddepth = cv2.CV_16S
3    kernel_size = 3
4
5    delta_i = cv2.Laplacian(image_i, ddepth, ksize=kernel_size)
6    delta_f = cv2.Laplacian(image_f, ddepth, ksize=kernel_size)
7
8    delta_i_mean = np.mean(delta_i)
9    delta_f_mean = np.mean(delta_f)
10
11   sigma_delta_i = delta_i - delta_i_mean
12   sigma_delta_f = delta_f - delta_f_mean
13
14   numerador = np.sum(sigma_delta_i * sigma_delta_f)
15   denominador = np.sum(np.power(sigma_delta_i, 2)) * np.sum(np.power
       (sigma_delta_f, 2))
16
17   epi =  numerador / np.sqrt(denominador)
18
19   return abs(epi)
```

Listing A.3: Implementación de la medida del índice de conservación de bordes

```
1  def gs(mu,sigma):
2    r1=random.random()
3    r2=random.random()
4
5    if r1<0.00001:
6      r1=0.00001
7    w1=math.sqrt(-2*math.log(r1))*math.cos(2*math.pi*r2)
```

```
  w2=math.sqrt(-2*math.log(r1))*math.sin(2*math.pi*r2)
  u=mu+w1*sigma
  v=mu+w2*sigma

  return u,v

for i in range(0,n-1):
      theta=((3*math.pi-Theta)/2)+(i*Theta/(n-1))
      for j in range(0,m-1):
        d=dmin+(j*(dmax-dmin)/(m-1))
        x=int(d*math.cos(theta)+w/2)
        y=int(-d*math.sin(theta))
        ip[y,x]=i0[y,x]

        AR=math.sqrt(ip[y,x])
        AI=0
        M=random.randrange(a,b)

        for k in range(1,10):#originalmente M
          (u,v)=gs(mu,sigma)
          AR=AR+u
          AI=AI+v
          Is[y,x]=AR**2+AI**2

        ir[j,i]=Is[y,x]
```

Listing A.4: Implementación del pseudocódigo para la generación del ruido speckle

```
# Se lee la carpeta de drive donde se encuentra el dataset
PATH = "/content/drive/MyDrive/Ultrasound/Dataset"

# Se lee las imagenes "originales"
PATH_INPUT = PATH + '/Dataset_input'
# Se lee las imagenes "objetivo"
PATH_TARGET = PATH + '/Dataset_target'

n = len(imgurls)
```

```
# Se toma el 80% del dataset para entrenamiento
train_n =round(n * 0.80)
# Se pasan los datos a typo numpy
randurls = np.copy(imgurls)
# Se barajean el dataset
np.random.shuffle(randurls)

# Se obtiene el conjunto de entrenamiento y prueba
tr_urls = randurls[:train_n]
ts_urls = randurls[train_n:n]

print(len(imgurls), len(tr_urls), len(ts_urls))

# Modifica el tama o de las imagenes a 256x256
def resize(input_image, real_image, height, width):
  input_image = tf.image.resize(input_image, [height, width],
                                method=tf.image.ResizeMethod.
    NEAREST_NEIGHBOR)
  real_image = tf.image.resize(real_image, [height, width],
                               method=tf.image.ResizeMethod.
    NEAREST_NEIGHBOR)

  return input_image, real_image

# Normaliza las imagenes a [-1, 1]
def normalize(input_image, real_image):
  input_image = (input_image / 127.5) - 1
  real_image = (real_image / 127.5) - 1

  return input_image, real_image

# Se usa una tuberia para los datos de prueba
test_dataset = tf.data.Dataset.from_tensor_slices(ts_urls)
test_dataset = test_dataset.map(load_image,
                                num_parallel_calls=tf.data.
    AUTOTUNE)
```

```
test_dataset = test_dataset.batch(BATCH_SIZE)

# Se usa una tuberia para los datos de entrenamiento
train_dataset = tf.data.Dataset.from_tensor_slices(tr_urls)
train_dataset = train_dataset.map(load_image,
                                  num_parallel_calls=tf.data.
    AUTOTUNE)
train_dataset = train_dataset.batch(BATCH_SIZE)
```

Listing A.5: Preprocesamiento del conjunto de datos basado en el codigo de TensorFlow (s.f.)

```
# Bloque downsample ENCODER
def downsample(filters, size, apply_batchnorm=True):
  # Inizialicion de pesos con ruido Gaussiano 0.02 (6.2)
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
      # Capa Convolucional 2D
      # filters = numero de filtros
      # size = tama o del filtro
      tf.keras.layers.Conv2D(filters, size, strides=2, padding='same
    ',
                             kernel_initializer=initializer,
    use_bias=False))

  if apply_batchnorm:
    # Capa de BachtNorm
    result.add(tf.keras.layers.BatchNormalization())

  # Capa de activacion LRELU
  result.add(tf.keras.layers.LeakyReLU())

  return result
```

Listing A.6: Bloque del codificador basado en el codigo de TensorFlow (s.f.)

```
# Bloque upsample DECODER
def upsample(filters, size, apply_dropout=False):
  # Inizialicion de pesos con ruido Gaussiano 0.02 (6.2)
  initializer = tf.random_normal_initializer(0., 0.02)

  result = tf.keras.Sequential()
  result.add(
    # Capa Convolucional Inversa 2D
    # filters = numero de filtros
    # size = tama o del filtro
    tf.keras.layers.Conv2DTranspose(filters, size, strides=2,
                                    padding='same',
                                    kernel_initializer=initializer,
                                    use_bias=False))

  result.add(tf.keras.layers.BatchNormalization())

  if apply_dropout:
      # Capa de Dropout
      result.add(tf.keras.layers.Dropout(0.5))

  # Capa de activacion RELU
  result.add(tf.keras.layers.ReLU())

  return result
```

Listing A.7: Bloque del decodificador basado en el codigo de TensorFlow (s.f.)

```
def Generator():
  inputs = tf.keras.layers.Input(shape=[256,256,3])

  down_stack = [
    downsample(64, 4, apply_batchnorm=False),
    # (bs, 128, 128, 64)
    downsample(128, 4),
    # (bs, 64, 64, 128)
    downsample(256, 4),
```

```
    # (bs, 32, 32, 256)
    downsample(512, 4),
    # (bs, 16, 16, 512)
    downsample(512, 4),
    # (bs, 8, 8, 512)
    downsample(512, 4),
    # (bs, 4, 4, 512)
    downsample(512, 4),
    # (bs, 2, 2, 512)
    downsample(512, 4),
    # (bs, 1, 1, 512)
  ]
  up_stack = [
    upsample(512, 4, apply_dropout=True),
    # (bs, 2, 2, 512)
    upsample(512, 4, apply_dropout=True),
    # (bs, 4, 4, 512)
    upsample(512, 4, apply_dropout=True),
    # (bs, 8, 8, 512)
    upsample(512, 4),
    # (bs, 16, 16, 512)
    upsample(256, 4),
    # (bs, 32, 32, 256)
    upsample(128, 4),
    # (bs, 64, 64, 128)
    upsample(64, 4),
    # (bs, 128, 128, 64)
  ]
  # Inizialicion de pesos con ruido Gaussiano 0.02 (6.2)
  initializer = tf.random_normal_initializer(0., 0.02)
  # Ultima Capa de la RED
  last = tf.keras.layers.Conv2DTranspose(
    3,
    4,
    strides=2,
    padding='same',
```

```
    kernel_initializer=initializer,
    activation='tanh') # (bs, 256, 256, 3)

  # Entradas
  x = inputs

  # Conecta las capas del ENCODER
  skips = []
  for down in down_stack:
    x = down(x)
    skips.append(x)

  # Eliminamos cueyo de botella
  skips = reversed(skips[:-1])

  # Conecta las capas del DECODER y las skip conections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    x = tf.keras.layers.Concatenate()([x, skip])

  # Conecta la ultima capa
  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

Listing A.8: Generador de tipo U-Net basado en el codigo de TensorFlow (s.f.)

```
# PATCHGAN
def Discriminator():
  # Inizialicion de pesos con ruido Gaussiano 0.02 (6.2)
  initializer = tf.random_normal_initializer(0., 0.02)

  # Entrada INPUT
  inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image
   ')
  # Entrada OBJETIVO
  tar = tf.keras.layers.Input(shape=[256, 256, 3], name='
```

```
  target_image')

# Concatena las dos imagenes
x = tf.keras.layers.concatenate([inp, tar])
# (bs, 256, 256, channels*2)
# Capas del Discriminador
down1 = downsample(64, 4, False)(x)
# (bs, 128, 128, 64)
down2 = downsample(128, 4)(down1)
# (bs, 64, 64, 128)
down3 = downsample(256, 4)(down2)
# (bs, 32, 32, 256)
# Capa ZeroPading
zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)
# (bs, 34, 34, 256)
# Capa Convolucional
conv = tf.keras.layers.Conv2D(
  512, 4, strides=1,
  kernel_initializer=initializer,
  use_bias=False)(zero_pad1) # (bs, 31, 31, 512)
# Capa de BachtNorm
batchnorm1 = tf.keras.layers.BatchNormalization()(conv)
# Capa de activacion LRELU
leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)
# Capa de zeropadiing
zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)
# (bs, 33, 33, 512)
# Ultima capa
last = tf.keras.layers.Conv2D(
  1,
  4,
  strides=1,
  kernel_initializer=initializer)(zero_pad2)
  # (bs, 30, 30, 1)

```

```
  return tf.keras.Model(inputs=[inp, tar], outputs=last)
```

Listing A.9: Implementación del discriminador basado en el codigo de TensorFlow (s.f.)

```
def discriminator_loss(disc_real_output, disc_generated_output):

  # Diferencia entre los true por ser real y el detectado por el
   discriminador
  # Que tan lejos estas de la imagen real
  real_loss = loss_object(tf.ones_like(disc_real_output),
   disc_real_output)

  # Diferencia entre los false por ser generado y el detectado por
   el discriminador
  # Que tan serca estas de la imagen generada
  generated_loss = loss_object(tf.zeros_like(disc_generated_output),
    disc_generated_output)

  total_disc_loss = real_loss + generated_loss

  return total_disc_loss

def generator_loss(disc_generated_output, gen_output, target):

  # Eror adversario
  # Imagen verdadera? test del discriminador
  gan_loss = loss_object(tf.ones_like(disc_generated_output),
   disc_generated_output)

  # mean absolute error
  l1_loss = tf.reduce_mean(tf.abs(target - gen_output))
  # Formula de la perdida del paper
  total_gen_loss = gan_loss + (LAMBDA * l1_loss)

  return total_gen_loss, gan_loss, l1_loss
```

Listing A.10: Implementación de las funciones de perdida para el generador y el discriminador basado en el codigo de TensorFlow (s.f.)

```
# Optimizadores de las 2 REDES
generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
```

Listing A.11: Implementación del optimizador basado en el codigo de TensorFlow (s.f.)

```
# Recorta aleatoriamente las imagens
def random_crop(input_image, real_image):
  stacked_image = tf.stack([input_image, real_image], axis=0)
  cropped_image = tf.image.random_crop(
      stacked_image, size=[2, IMG_HEIGHT, IMG_WIDTH, 3])

  return cropped_image[0], cropped_image[1]

# Aumenta el Dataset mueve el centro y giro aleatorio
@tf.function()
def random_jitter(input_image, real_image):
  # resizing to 158 x 158 x 3
  input_image, real_image = resize(input_image, real_image, 158,
   158)

  # randomly cropping to 128 x 128 x 3
  input_image, real_image = random_crop(input_image, real_image)

  if tf.random.uniform(()) > 0.5:
    # Voltear aleatorio
    input_image = tf.image.flip_left_right(input_image)
    real_image = tf.image.flip_left_right(real_image)

  return input_image, real_image
```

Listing A.12: Aumento sintético del conjunto de datos basado en el codigo de TensorFlow (s.f.)

```
# Bloque downsample ENCODER
def downsample(filters, size):
  # Inizialicion de pesos con ruido Gaussiano 0.02 (6.2)
  initializer = tf.random_normal_initializer(0., 0.02)
```

```
77
78    result = tf.keras.Sequential()
79    result.add(
80        # Capa Convolucional 2D
81        # filters = numero de filtros
82        # size = tama o del filtro
83        tf.keras.layers.Conv2D(filters, size, strides=2, padding='same
      ',
84                               kernel_initializer=initializer,
      use_bias=False))
85    # Capa de activacion RELU
86    result.add(tf.keras.layers.ReLU())
87
88    return result
```

Listing A.13: Bloque del codificador modificado basado en el codigo de TensorFlow (s.f.)

```
89  # Bloque upsample DECODER
90  def upsample(filters, size):
91    # Inizialicion de pesos con ruido Gaussiano 0.02 (6.2)
92    initializer = tf.random_normal_initializer(0., 0.02)
93
94    result = tf.keras.Sequential()
95    result.add(
96      # Capa Convolucional Inversa 2D
97      # filters = numero de filtros
98      # size = tama o del filtro
99      tf.keras.layers.Conv2DTranspose(filters, size, strides=1,
100                                      padding='same',
101                                      kernel_initializer=initializer,
102                                      use_bias=False))
103   # Capa de activacion LRELU
104   result.add(tf.keras.layers.LeakyReLU())
105   # Capa de Dropout
106   result.add(tf.keras.layers.Dropout(0.5))
107   # Capa de BatchNormalization
108   result.add(tf.keras.layers.BatchNormalization())
```

```
109
110   return result
```

Listing A.14: Bloque del decodificador modificado basado en el codigo de TensorFlow (s.f.)

```
111 def Generator():
112   inputs = tf.keras.layers.Input(shape=[128,128,3])
113
114   down_stack = [
115     downsample(16, 3),  # (bs, 64, 64, 16)
116     downsample(32, 3),  # (bs, 32, 32, 32)
117     downsample(64, 3),  # (bs, 16, 16, 64)
118     downsample(64, 3),  # (bs, 8, 8, 64)
119     downsample(64, 3),  # (bs, 4, 4, 64)
120   ]
121   up_stack = [
122     upsample(64, 3), # (bs, 8, 8, 64)
123     upsample(32, 3), # (bs, 16, 16, 32)
124     upsample(32, 3), # (bs, 32, 32, 32)
125     upsample(16, 3), # (bs, 64, 64, 16)
126     upsample(1, 3),  # (bs, 128, 128, 1)
127   ]
128   # Inizialicion de pesos con ruido Gaussiano 0.02 (6.2)
129   initializer = tf.random_normal_initializer(0., 0.02)
130   # Ultima Capa de la RED
131   last = tf.keras.layers.Conv2DTranspose(
132      3,
133      1,
134      strides=1,
135      padding='same',
136      kernel_initializer=initializer,
137      activation='tanh') # (bs, 128, 128,
138      3)
139
140   # Entradas
141   x = inputs
142
```

```
143   # Conecta las capas del ENCODER
144   skips = []
145   for down in down_stack:
146     x = down(x)
147     skips.append(x)
148 
149   # Eliminamos cueyo de botella
150   skips = reversed(skips[:-1])
151 
152   # Conecta las capas del DECODER y las skip conections
153   for up, skip in zip(up_stack, skips):
154     x = up(x)
155     x = tf.keras.layers.Concatenate()([x, skip])
156 
157   # Conecta la ultima capa
158   x = last(x)
159 
160   return tf.keras.Model(inputs=inputs, outputs=x)
```

Listing A.15: Generador modificado basado en el codigo de TensorFlow (s.f.)

```
161 def Discriminator():
162   # Inizialicion de pesos con ruido Gaussiano 0.02 (6.2)
163   initializer = tf.random_normal_initializer(0., 0.02)
164 
165   # Entrada INPUT
166   inp = tf.keras.layers.Input(shape=[128, 128, 3], name='input_image
       ')
167   # Entrada OBJETIVO
168   tar = tf.keras.layers.Input(shape=[128, 128, 3], name='
       target_image')
169 
170   # Concatena las dos imagenes
171   x = tf.keras.layers.concatenate([inp, tar]) # (bs, 128, 128, 6)
172   # Capas del Discriminador
173   down1 = downsample(16, 3)(x)      # (bs, 64, 64, 16)
174   down2 = downsample(32, 3)(down1)  # (bs, 32, 32, 32)
```

```
175   # Capa ZeroPading
176   zero_pad1 = tf.keras.layers.ZeroPadding2D()(down2)
177   # (bs, 34, 34, 32)
178   # Capa Convolucional
179   conv1 = tf.keras.layers.Conv2D(
180     32,
181     3,
182     strides=1,
183     kernel_initializer=initializer,
184     use_bias=False)(zero_pad1)
185     # (bs, 31, 31, 32)
186   # Capa de BachtNorm
187   batchnorm1 = tf.keras.layers.BatchNormalization()(conv1)
188   # (bs, 31, 31, 32)
189   # Capa de activacion LRELU
190   leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)
191   # Capa de zeropadiing
192   zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)
193   # (bs, 33, 33, 32)
194   # Capa Convolucional
195   conv2 = tf.keras.layers.Conv2D(
196     32, 3, strides=1,
197     kernel_initializer=initializer)(zero_pad2)
198     # (bs, 31, 31, 32)
199   # Capa final
200   last = tf.keras.layers.Conv2D(
201     1, 1, strides=1,
202     kernel_initializer=initializer)(conv2)
203     # (bs, 31, 31, 1)
204
205   return tf.keras.Model(inputs=[inp, tar], outputs=last)
```

Listing A.16: Discriminador modificado basado en el codigo de TensorFlow (s.f.)